Disk Operating System
Instructional and Reference Manual

Do's and Don'ts
of DOS

NOTICE

Apple Computer Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

APPLE COMPUTER INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL OR WITH RESPECT TO THE SOFTWARE DESCRIBED IN THIS MANUAL, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. APPLE COMPUTER INC. SOFTWARE IS SOLD OR LICENSED "AS IS". THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING THEIR PURCHASE, THE BUYER (AND NOT APPLE COMPUTER INC., ITS DISTRIBUTOR, OR ITS RETAILER) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL APPLE COMPUTER INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE, EVEN IF APPLE COMPUTER INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

Reorder APPLE Product #A2L0012
(030-0011-01)

THE DO'S AND DON'TS OF DOS

A MANUAL FOR USING THE APPLE DISK II WITH DOS VERSION 3.2

If many faultes in this book you fynde,
Yet think not the correctors blynde;
If Argos heere hymselfe had beene
He should perchance not all have seene.

Richard Shecklock...1565

Written by Phyllis Cole and Brian Howard
with lots of help from their friends,
and some hindrance from the subject matter,
which kept going 'round and 'round.

TABLE OF CONTENTS

CHAPTER 3

EXERCISING OPTIONS

CHAPTER 4

PLAYING SAFE

CHAPTER 5

MORE "HOUSEKEEPING" INFORMATION

CHAPTER 6

USING SEQUENTIAL FILES

CHAPTER 7

AUTO APPLE

CHAPTER 8

USING RANDOM-ACCESS FILES

CHAPTER 9
USING MACHINE LANGUAGE FILES

CHAPTER 10
INPUT, OUTPUT AND CHAINING

APPENDIX A
FILE TYPES USED WITH DOS COMMANDS

APPENDIX B
DOS MESSAGES

APPENDIX C

FORMAT OF DISKETTE INFORMATION

APPENDIX D

MEMORY USAGE

APPENDIX E

DOS ENTRY POINTS AND SCHEMATICS

APPENDIX F
SUMMARY OF DOS COMMANDS

APPENDIX G
SUMMARY OF DOS PROCEDURES

INDICES

PREFACE

This manual has two primary functions. The first is to teach you how to use the DOS (Disk Operating System); the Chapters of the manual use examples to accompany explanations of how the various DOS commands work. The second function of the manual is to serve as a reference guide to DOS. The Appendices, Quick Reference Card, and the Indices (on pages 172, 178 and the inside back cover) were planned with this function in mind.

To use an Apple Disk II, you need an Apple II computer with at least 16K of memory -- but 32K is recommended, since the 16K system allows little memory space to store programs. For using Apple Disk II with Applesoft BASIC on the firmware ROM card (Part A2B0##9X), your computer still requires only 16K of memory. For using Apple Disk II with Applesoft on cassette tape (Part A2T0004) or on diskette, your computer must have at least 32K of memory.

The Apple Disk II is a "floppy" disk unit which allows you to store and retrieve information much more quickly and conveniently than you can with tape. The information is stored and retrieved from a "diskette", a small (about 5-inch diameter) specially coated plastic disk which is permanently sealed in a square plastic case.

One of the most important advantages to using Disk II is that information is stored and retrieved by a name under which it is filed. A program that catalogs phone numbers might be saved with an instruction such as
SAVE PHONE NUMBERS
and retrieved with an equally simple command. The name PHONE NUMBERS under which the program is filed is a file name.

The programs that automatically keep track of files, save and retrieve information, and do a multitude of other housekeeping tasks are called the Disk Operating System, usually shortened to "DOS". Some people say "doss" and others say "dee oh ess". Learning to use DOS and the disk consists of learning a few special DOS commands described in this manual. These commands can be used as extensions to either Applesoft or Integer BASIC or machine language programs.

At some places you'll see the symbol

preceding a paragraph. This symbol indicates an unusual feature to which you should be alert.

The symbol

precedes paragraphs describing situations from which BASIC may be unable to recover. You will lose your program, probably have to re-start DOS, and may have to re-start BASIC.

*** NOTE ***

This manual applies to DOS version 3.2 only. and descriptions may not be correct for DOS versions 3.1 and 3.0. If you do not have DOS version 3.2, you should get a copy of it from your dealer before using this manual. The version number of DOS is shown when you boot the System Master diskette. DOS version 3.2 is on a floppy diskette, Part 804-0002-03. The UPDATE program, discussed in Chapter 5, can be used to convert disks with outdated versions of DOS to DOS version 3.2.

CHAPTER 1
INSTALLATION AND HANDLING

UNPACKING

Your Disk II system consists of seven items:

1) The disk drive (the main box).
2) A printed-circuit card (the controller card) that plugs into the Apple II.
3) A flat ribbon cable, already fastened to the disk drive, for connecting the disk drive to the controller card.
4) A "SYSTEM MASTER" diskette.
5) A blank diskette.
6) A warranty card.
7) Adhesive labels.
8) This manual.

If you have purchased a drive only (for example, as a second drive for your controller card) your system will include all of the above items except 2), the controller card.

Save the packing material in case you wish to transport your disk -- or in the unlikely event you must return it to your dealer or to the factory for service. Send in the warranty card -- not only does this put the warranty in effect, but it puts you on the mailing list for CONTACT, the Apple users' newsletter that keeps you informed of updates and new products.

*** Special Note ***

Before connecting or disconnecting
ANYTHING
on the Disk II or Apple II
TURN OFF THE POWER.
This is a must.

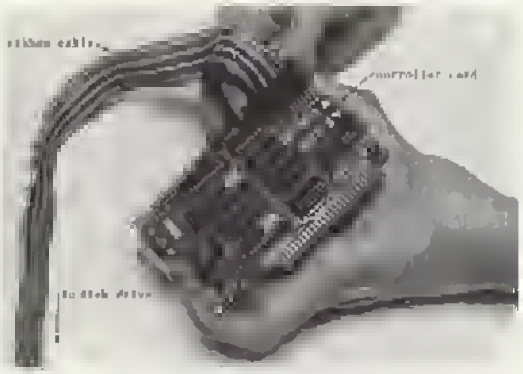
CONNECTING THE CABLE

In use, the disk drive will be connected to the controller card by the flat, ribbon-like cable. One end of this ribbon cable is already fastened to the disk drive. If this is your first disk drive, the connector at the end of the ribbon cable from this drive should be attached to the upper set of pins on the controller card. This set of controller card pins is labelled "DRIVE 1".

*** Caution ***

If the cable from the disk drive to the controller card is not plugged into the controller card correctly, considerable physical damage can be done to the disk drive unit and its electronics. To assure correct assembly, be sure to plug the ribbon cable into the controller card before installing the controller card into the computer. Two installation tips follow. First, don't jam the cable between the connector and the controller card. When the cable is plugged into the controller card correctly, the cable should exit from its connector on the side of the connector that is away from the controller card, as

shown in the photograph. Second, make sure that all the pins of the controller card's connector go into the matching holes in the ribbon cable's connector. By making the connection before installing the card, you can actually see that all the pins are going into the holes correctly.



Connecting the Cable to the Controller

If you are installing a second disk drive, you should connect the ribbon cable from the second drive to the lower set of pins on your controller. This set of pins is labeled "DRIVE 2". Take the same care attaching this connector as you did with the first.

INSTALLING THE CONTROLLER

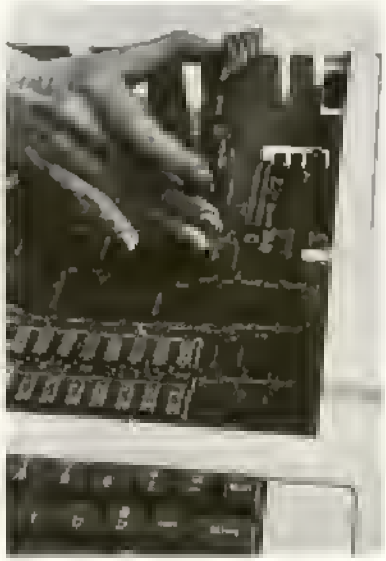
To install the Disk II controller card, which you have already connected to the disk drive via the ribbon cable, you will simply plug the controller card into a socket inside the Apple II, as follows:

1. Turn off the power switch at the back of the Apple II. This is important to prevent damage to the computer. If the power is on, removal or insertion of any card could cause permanent damage to both the card and the Apple II.

2. Remove the cover from the Apple II. This is done by pulling up on the cover at the rear edge (the edge farthest from the keyboard) until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide the cover backward until it comes free.

3. Inside the Apple II, across the rear of the circuit board, there is a row of eight long, narrow sockets called "slots". The leftmost one (looking at the computer from the keyboard end) is slot #0, and the rightmost one is slot #7. Locate slot #6, one socket to the left of the rightmost socket. The controller card may be placed in any slot except slot #0, the leftmost. However, Apple's standard location for the disk controller card is slot #6, and most Apple software (and this manual) is written with that location in mind.

4. BE SURE THE POWER IS OFF BEFORE YOU INSERT OR REMOVE ANY CARD FROM THE COMPUTER. Insert the "fingers" portion of the controller into slot #6. The "fingers" portion will enter the socket with some friction and will then seat firmly. Since the fingers make electrical contact, it is a good idea to keep your fingers from touching them. Before installation, you may wish to use rubbing alcohol to clean the fingers on the board (and, optionally, your own fingers if you're so inclined).



Inserting the Controller Card

5. Adjust the ribbon cable so it lays flat and passes over one of the areas between the vertical openings in the back of the Apple II case, as shown in the drawing. When the lid is installed it will clamp down the cable and act as a strain relief.

Cable Placement

6. Replace the cover of the Apple II; remember to start by sliding the front edge of the cover into place. Press down on the two rear corners until they pop into place.

7. The Disk II controller is installed, and the Apple II may now be turned on. Place the disk drive in a convenient location, usually alongside of or on top of your Apple II.

INSTALLING MULTIPLE DISK DRIVES

Each controller card can be used with two disk drives, one attached to the upper set of pins, labeled "DRIVE 1", and the second attached to the lower set of pins, labeled "DRIVE 2". Your first disk drive should be attached to the DRIVE 1 pins and the second to DRIVE 2 pins on the card in slot #6. The third and fourth drives should be attached to the DRIVE 1 and DRIVE 2 pins, respectively, on a card in slot #5, the fifth and sixth drives attach to the DRIVE 1 and DRIVE 2 pins on a card in slot #4, and so on.

If you have multiple drives, it is a good idea to label the front of each drive with its slot and drive number since your programs will refer to the disks by those numbers. Adhesive labels are provided in an envelope with your drive.

CARE OF THE DISK II AND DISKETTES

The Disk II drive, unlike the Apple II, is a mechanical device, with motors and moving parts. Therefore it is somewhat more delicate than the computer. Rough handling, such as dropping the drive, or having things drop on it, can cause it to malfunction. The drive should not be placed beside or on a TV set, since the strong magnetic fields put out by TVs may cause damage to the magnetic properties of the drive. If in doubt, locate disk drives at least 2 feet from any TV set.

Each diskette is a small (about 5-inch diameter) plastic disk coated so that information may be stored on and erased from its surface. The coating is similar to the magnetic coating on recording tape. The diskette is permanently sealed in a square black plastic cover which protects it, helps keep it clean and allows it to spin freely. This package is never opened.

The term "floppy" comes from the fact that the diskette is flexible. Older computer information storage devices that worked on similar principles used rigid disks. While the diskette (and its plastic cover) are somewhat flexible, actually bending the diskette can damage it. The diskette cover contains both lubricants and cleaning agents to extend trouble free operation -- treat covers with respect.

Never let anything touch the brown or gray surface of the diskette itself. Handle the diskette by the black plastic cover only. When a diskette is not in use, keep it in the paper pocket that it came in. These pockets are creased to minimize static build-up which attracts dust. It is best to store diskettes vertically when they are not in use. Vinyl notebooks especially made for this purpose are convenient.

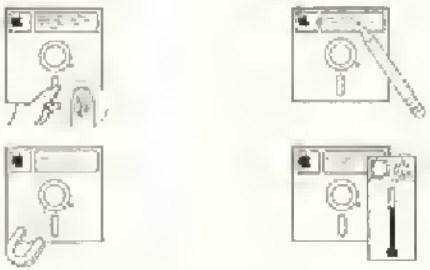
Diskettes hold a tremendous amount of information: a single diskette can hold over 931,#0# bits of information. An individual bit of information, therefore, occupies a very small portion of the diskette. An invisible scratch on the surface of the diskette, or even a fingerprint, can cause errors. Do not place diskettes on dirty or greasy surfaces; do not let them collect dust.

To write on a diskette label, use a FELT TIP pen. Do not press hard. It is best not to write on a label attached to a diskette, but to write on the separate label, then attach it to the diskette.

Keep diskettes away from magnetic fields. This means to keep them away from electric motors and magnets; they should not be placed on top of electronic devices such as television sets. They may be temporarily laid on the Apple II or the Disk II.

Diskettes are sensitive to extremes of temperature. Keep diskettes out of the sun, and away from other sources of heat that can cause them to warp and/or lose data. On hot days, car trunks (or dashboards) can be diskette killers. Diskettes operate satisfactorily up to 125 degrees Fahrenheit (51.7 Celsius), which is not very hot. The first evidence of heat damage is a warped or bent black plastic cover.

With reasonable care a diskette will give you an average life of 40 hours -- which is a lot, when you consider the few seconds it takes to LOAD most programs. With just a little bit of carelessness, a diskette may give you no service at all.



No-No's

INSERTING AND REMOVING DISKETTES

Using a disk drive is far quicker and easier than using a cassette recorder, however some care is necessary to protect the diskettes. The drive itself must also be handled with some care. The drive door is opened by pulling outward on its bottom edge. The diskette is then slipped into the slot with the label upwards, as shown in the photograph. The edge of the diskette with the oval cutout in the diskette's square plastic cover should enter the drive first. The edge of the diskette with the label should enter the drive last.

A Good
RULE OF THUMB

Hold a diskette with your right thumb over the label:
that pretty much insures the correct orientation
when you put the diskette in the drive.

Inserting a Diskette

Push the diskette gently until the diskette is entirely into the drive. Do not bend the diskette! If it is pushed in too hard, the diskette can be permanently damaged. Close the drive door by pushing it down again. The two metal fingers (which can be seen inside the slot when the drive door is closed) should just clear the diskette as the door closes.

A diskette is removed by opening the drive door and pulling the diskette carefully out of the drive. The act of opening the disk drive door lifts the "head" off the disk. If you plan to leave an unused diskette in a drive for several hours, it's a good idea to open the door so the head won't rest on the diskette.

STOP

NEVER remove a diskette while the drive's "IN USE" light is on. This may permanently damage the diskette, and is almost sure to destroy the information on it. In such a case, the diskette can usually be re-used, but you won't be able to recover the lost information.

∞

CHAPTER 2
GETTING STARTED

BACKGROUND

Learning to use the disk and its operating system consists of learning a few special instructions, several of which are straightforward extensions of familiar BASIC instructions. This manual assumes that you're familiar with the Apple II, and feel comfortable writing simple BASIC programs.

To learn how to use the Apple II and Integer BASIC, consult the Apple II BASIC Programming Manual (Apple Product #A2L0005X). To learn how to use Applesoft BASIC, consult the Applesoft II BASIC Programming Reference Manual (Apple Product #A2L0006). The Applesoft manual assumes you're already familiar with the Apple II and simple BASIC programming. If you're not familiar with either manual, we will wait here while you learn about the Apple II, before going on to learn about DOS.

*
*
*

Throughout the manual are listings of programs that illustrate how to use DOS. Most of these programs are in Applesoft; a few are in Integer BASIC. Sometimes the changes needed to convert an Applesoft program to Integer BASIC are mentioned; other times, they are not. Consult Appendix H in the Applesoft manual for details on the differences between programs written in Integer BASIC and Applesoft BASIC.

A little bit of hands-on experience is worth a lot of reading. Once your disk drive is hooked up and the computer is turned on, follow each of these descriptions by actually trying out the procedures on your Apple II.

Put the Apple II into BASIC -- either Integer BASIC or Applesoft. Place the System Master diskette into the drive. The diskette should be labelled 604-0002-XX. The last two digits are indicated by X's, since it doesn't matter what they are. If you have more than one drive, use Drive 1. This section of the manual only deals with one drive and assumes that you've followed the standard conventions, putting the controller into slot #6.

With the disk drive attached, and the diskette in the drive, and the disk drive door closed, you will find that the Apple II performs just as it did without the disk. Nothing is changed. It is as if the disk drive were not there. And, as far as the Apple II is concerned, the disk drive is not connected yet: a special command must be given to inform the computer that the disk drive and the new DOS instructions are available.

Even though DOS commands look like BASIC commands, they do not always follow the same rules. For example, multiple DOS commands cannot be put on one line, separated by commas. The SYNTAX ERROR message results.

SPECIAL KEYS

Sometimes this manual uses the curly brackets { and } to enclose the names of special keys which you are supposed to press on the Apple II keyboard.

{RETURN} means you should press the key marked "RETURN". Press the RETURN key after each instruction.

{RESET} means press the key marked "RESET". A press of the RESET key will put you into the MONITOR program, which uses * as its prompt character.

{ESC} means press the key marked "ESC". "ESC" originally meant "escape", but nowadays has other uses.

{CTRL} is a bit different. It means you should press the key marked "CTRL" (which stands for control) and continue holding it down while you type another key. For example, {CTRL}C means type the "C" key while you are holding down the CTRL key. Sometimes use of the control is indicated in another way: CTRL-C and {CTRL}C both mean the same thing.

*** NOTE ***

Characters typed while holding down the CTRL key do not appear on the screen.

BOOTING DOS

The process of adding the DOS commands to the BASIC in your Apple II is called booting the disk. The disk may be booted from Integer BASIC, from Applesoft or from the Monitor. There are various ways you can use to boot DOS. From Integer BASIC or Applesoft, the PR#s and IN#s commands (see your Applesoft manual) may be used. From the Monitor, "control commands" using the CTRL key may be used. Once you get DOS booted, it's all the same DOS: it doesn't matter how you got there.

In the examples below, the lower-case letter s stands for the number of the Apple II slot in which your disk controller card is located. The standard location for the controller card is slot #6 (see Chapter 1, Installing the Controller). After any of the following commands, you must press the RETURN key.

From Integer BASIC (whose prompt character is >)
you can use either of these commands to boot the disk:

You type:	PR#s	Example: PR#6
or:	IN#s	Example: IN#6

From Applesoft (whose prompt character is])
you can use either of these commands to boot the disk:

You type:	PR#s	Example: PR#6
or:	IN#s	Example: IN#6

From the Monitor (whose prompt character is *),
you can use any of these commands to boot the disk:

You type:	Cs00G	Example: C600G
or:	s{CTRL}K	Example: 6{CTRL}K
or:	s{CTRL}P	Example: 6{CTRL}P

In the rest of this manual, when you are to re-start the DOS in this manner we will simply say: "boot the DOS" or "boot the disk". Both expressions (very popular among computer users) mean the same thing. "Boot" is short for the word "bootstrap" and the term is from the expression "to pull oneself up by one's bootstraps;" but a more thorough etymology would be out of place here. In any case, it does not mean to kick the disk, even if you do feel in such a mood from time to time.

Now try booting DOS from your System Master diskette. Start by putting your Apple II in BASIC -- either Integer BASIC or Applesoft will do. Be sure the diskette is properly inserted. Next type

```
PR#6
```

and press the RETURN key as usual. From now on, it will be assumed that you will press the RETURN key after each instruction.

Once you press the RETURN key, the red "IN USE" light will come on, the disk will make whirring and clacking noises (don't be alarmed -- it's not getting ready to fly away) and in less than 10 seconds, a message will appear. The message should be similar to the following:

```
DISK II MASTER DISKETTE  VERSION 3.4
                              16-FEB-79
COPYRIGHT 1979  APPLE COMPUTER INC
```

If you now try to use BASIC, you will find that most commands still operate normally and, aside from the message suddenly appearing, the Apple II seems unchanged. What has happened is this: a few new commands have been introduced, and a few old ones have new capabilities. Two changes have been made that are not obvious, however:

1) The HIMEM pointer to the highest memory location you may use has been reset to accomodate the DOS program.
2) Your Apple II may have lost some of its high-resolution graphics capabilities, depending on the amount of memory in your computer.

For details, see Appendix D on DOS Memory Usage.

IF BOOTING DOESN'T WORK

If you can't successfully boot your System Master diskette, re-read the manual carefully -- that cures 90% of all problems.

This isn't likely, but if your unit was shipped in a Sherman tank or some such, the connectors inside the disk drive may have worked a bit loose. If you are at all squeamish about handling the insides of your drive, your dealer will be glad to check it out.

If you enjoy getting your fingers into the works, you can turn the computer off, and disconnect the drive from the controller. Loosening the four screws on the bottom of the drive allows the mechanism to slip forward out of the case. Tighten the connectors by pushing them gently onto the circuit boards. Re-assemble the unit and it will probably now work. If this first aid doesn't work, see your dealer. Don't make any adjustments.

INITIALIZING NEW DISKETTES

The System Master that comes with this manual is a very special diskette: it contains programs that allow you to copy an entire diskette (if you have two disk drives), update any diskette that has an earlier version of DOS, and more. Programs that demonstrate various capabilities of DOS are also included on the diskette and discussed in the manual.

Take the System Master diskette from the drive, and replace it with the other blank diskette supplied with your drive. Now try an experiment. Get BASIC going, then type

```
PR#6
```

and watch what happens. The red IN USE light comes on, and the disk drive makes a few clackety noises, then it just keeps whirring softly and quietly and it doesn't stop. You'll have to press the RESET key to stop it (normally, this is a BAD idea, but these circumstances aren't normal). It's a good idea to open the disk drive door before pressing the RESET key, since that lifts the head of the disk drive off the surface of the diskette.

What happened was this: your Apple II went on a fruitless unending search for information on a blank diskette (on a clear disk you can seek forever...). When a new diskette is manufactured, it contains no information at all, like a blank tape purchased for a tape recorder. To operate in the computer, there must be special information placed on the diskette: the diskette must be initialized.

If you've been keeping up with the hands-on part of the example, your blank diskette is in the drive and you just pressed the RESET key. Now take out the blank diskette, replace it by the System Master diskette, and close the door of the drive. Get the computer into BASIC and type

```
PR#6
```

again. You should again get the message you got before when you booted. Once more the DOS commands have been added to BASIC.

The INIT command can be used to INITialize a "slave" diskette. Slave diskettes are memory-size dependent: the size of a system on which a diskette is INITialized determines what size system can use the disk. If a slave diskette is created on a 32K system, then it can be used only on 32K systems or larger. But on larger systems, only 32K of memory will be used. After INITializing a slave diskette, you can use the UPDATE 3.2 program (see Chapter 5) to transform your slave diskette into a "master" diskette whose DOS is self-relocating so that memory is used efficiently.

The INIT command requires the use of a BASIC program called the "greeting" program since it greets you: each time you boot the diskette the program will be run automatically. The greeting program is commonly named "HELLO" but you could call it "BONJOUR" or "BUENOS DIAS" or whatever you like. It helps keep life simple to use a standard name for greeting programs as you INITialize diskettes.

Here's a step-by-step guide to INITializing a slave diskette. We assume DOS is already booted as described above.

1) Remove the System Master from your drive and replace it with a blank diskette.

2) Type NEW, then type a greeting program. Here is a simple sample of a greeting program:

```
5  REM   GREETING-t PROGRAM
10  PRINT "SLAVE DISKETTE CREATED
      ON 32K SYSTEM"
20  PRINT "BY MMY DUNKS ON 8 AUGU
     ST 1980"
30  END
```

You should supply your own name, system size, the current date and other information to help you quickly and easily determine the diskette's history and slave/master status. You may RUN the program to see if it does what you expect.

3) Once the program is satisfactory, type this instruction:
INIT HELLO
When you press the RETURN key, the diskette will spin for nearly two minutes, making clacks and little whispery noises every now and then. The appropriate prompt character (e.g.] for Applesoft) will be displayed when INITialization is complete.

4) When the disk quiets down and the IN USE light goes off, remove the diskette and label it. The label should say something like
32K SLAVE DISKETTE CREATED 8 AUGUST 1980
so that just by looking at it you know it isn't blank.

Put aside the System Master diskette supplied by Apple Computer. Put it where it won't be damaged by heat, physical stress (kids? dogs?) or magnetic objects. And where it won't get lost. It should be treated especially carefully, since it contains many useful programs.

Once a diskette has been INITialized, it will be referred to as a slave diskette. To label your slave diskette, you had to take it out of the drive. Put it back in and try booting it: the message in your PRINT statements should appear. If you followed the model given above, the screen should say:

```
SLAVE DISKETTE CREATED ON 32K SYSTEM
BY AMY DUNKS ON 8 AUGUST 1980
```

Since the once-blank diskette now can boot, you know that it has been INITialized correctly. From this point on you will use the newly INITialized slave diskette for experimentation. You cannot do some of the procedures to be demonstrated below on the System Master, because the diskette is "write protected", as discussed in Chapter 4.

If you have purchased additional blank diskettes, it would be a good idea to INITialize a few of them now.

LOAD-ING AND SAVE-ING WITH DOS

Boot the system with your initialized diskette. Type
NEW
to make sure no programs are in memory. This will erase your greeting program (which is LOADed and RUN when you boot DOS) from memory (but not from the diskette).

Now type this simple program:

```
5  REM   COUNT PROGRAM
10  FOR I = 1 TO 10
20  PRINT I;
30  NEXT I
40  END
```

RUN it once or twice to make sure that it works as you expect. In Applesoft, when the program is RUN you'll see this:

```
1              2              3
4              5              6
7              8              9
10
```

For reference purposes, call this program ONE TO TEN, since it counts from one to ten. To store this program on the diskette, type the instruction
SAVE ONE TO TEN
When you complete the command by pressing the RETURN key, the disk will whirr for a few seconds, and the program will be saved.

If you had typed
SAVE
without any name, the program would have been saved on cassette tape, as usual (assuming you had operated the tape recorder as described in the BASIC Programming Manual).

To prove that the program has been SAVEd on diskette, do the following.
First, type
LIST
then
RUN
to see the program is still in memory and still operates properly. This demonstrates that using DOS to SAVE a program on a diskette doesn't affect the program in any way.

Now type
NEW
then
LIST
There will be no program left at all -- it disappeared when you typed NEW. To really make sure the program is dead, turn off the computer. You can even take the diskette out and put it (gently) back in again. Turn the computer back on again, get into BASIC and boot the DOS. Type NEW (which erases the HELLO program), and then LIST. Nothing there? Right.

Now type
```
LOAD ONE TO TEN
```
and the disk will whirr for about two seconds. LIST the program: it is revived. RUN it, and you will find it in perfect health. That is all there is to SAVEing and LOADing programs from disk: it's just like using the cassette tape except that a file name is used, and it's faster.

CATALOG

You stored the program ONE TO TEN on your diskette. Actually, you had already stored another program. To see what programs are stored on a given disk, type the command
```
CATALOG
```
and a list of all the programs on the diskette will appear. Right now your diskette's catalog should look like this, if your programs were written in Integer BASIC:
```
 I 002 HELLO
 I 002 ONE TO TEN
```

The letter "I" in the left column means that the programs are in Integer BASIC; before names of Applesoft programs you'll see an "A". Besides BASIC program files, there are also other kinds of files that can be stored, and they will be explained in Chapters 6 through 9. The numbers after the file-type letter represent the length of the stored program. In this case, 002 diskette "sectors" were required to store the programs. Each diskette sector can store up to 256 bytes of information. The shortest possible file, an empty text file (see Chapter 6), requires 001 sector to record certain "housekeeping" information. In all, a diskette can store 403 sectors of programs and other files. Lastly, each entry in the catalog contains the name of the program. See Appendix C for details on how information is stored on the diskettes.

When a file exceeds 255 sectors, the length reported for that file by CATALOG starts over again at 000.

There is no way to tell from looking at the CATALOG which program is the greeting program. So it helps if you always give the same name to your greeting program. To change the name of the greeting program, see the discussion of the UPDATE program, in Chapter 5.

Sometimes you'll have more programs on a diskette than will fit on the TV screen at one time. CATALOG will cause the first 18 programs to be listed. When you're ready to see the other programs on the diskette, press any key except the RESET key, CTRL key or the SHIFT keys.

WHAT'S IN A NAME?

File names must be from 1 to 30 characters in length; DOS will truncate longer file names to 30 characters. A file name must begin with a letter. Any typeable character, except the comma (,) may appear in the name.

Here are some legal file names:

```
SOMNAMBULISTICS
ONE TO TEN
HIRES 34
THE QUALITY OF MERCY: UNSTRAINED
```

Here a few names that will not work (and reasons why):

```
1 TO 10                                 (begins with a digit)
HI THERE, BABE                          (contains a comma)
INEPT EXCESS VERBIAGE DISQUALIFIES NAMES  (will be cut to 30 characters)
```

Although the name of the last file will be cut to 30 characters when displayed by CATALOG, you can, if your fingers can take it, type the entire name when LOADing or RUNning, and all will work correctly.

Every line in the catalog represents a "file". The BASIC program you stored is an example of a file. The rules given here for file names also apply to the names of programs.

If a control character is accidentally (or even purposefully) typed into a name, that character will not appear on the screen when you get a catalog. For example, if you type (CTRL)T instead of plain "T" in the name "AGATHA", the catalog listing would appear to be

```
AGAHA
```

However, if you tried to LOAD that file by typing

```
LOAD AGAHA
```

the computer would reply

```
FILE NOT FOUND
```

even though the name you typed seemed to be identical to the name in the catalog. So be careful: don't inadvertently put control characters in file names. (Although, heh heh, it's a clever way to keep you out of my bank records if all files have secret control characters embedded in them...) The File Names section of Appendix F contains tips on how to find out what control characters are imbedded in file names.

RENAME-ING FILES

For one reason or another, you'll occasionally want to change the name of a file. Suppose you get tired of typing the file name ONE TO TEN and decide to call the file COUNT. Just type

```
RENAME ONE TO TEN, COUNT
```

and after a moment of whirring you'll again see the BASIC prompt character. Now type

```
CATALOG
```

to verify that all went as planned.

The RENAME command does not check to see whether the new name you're using already exists or not, so it's entirely possible that you can RENAME until all files on a diskette have the same name...a most undesirable and confusing situation that is best avoided.

DELETE-ING FILES

It is easy to remove files from the diskette. Type
CATALOG
again to see the two files that are on your diskette. Now type
LOAD COUNT
(assuming you changed the file name as shown above) to get that program into memory. Delete this program from the diskette by the instruction
DELETE COUNT
and test that your deletion has worked by typing
CATALOG

Only the greeting program -- probably called HELLO -- is left. Since the program COUNT is in memory (that's why you LOADed it), you can place it back onto the diskette with the familiar command
SAVE COUNT
Take a look at the catalog to see that the program is again on the diskette.

If you try to DELETE a file that's not on the diskette, you'll receive the
FILE NOT FOUND
message.

RECOVERING FROM ACCIDENTAL RESETS

Suppose you're without the DOS in either Integer BASIC or Applesoft. (If Applesoft is in firmware, we assume the switch on the card is set for Applesoft). If you accidentally strike the RESET key, you can recover with your program intact by using CTRL-C. The DOS also has recovery procedures that will usually preserve your program and data.

If you have already booted DOS, and then press RESET, you get the Monitor (*) prompt. To return to DOS and the BASIC you left, type
3D0G
Remember that's a zero, not the letter O, between the letters D and G.

If recovery to DOS does not work, and the program still LISTs, all is not lost: save the program on tape (you did remember to keep your tape drive for just such an emergency, didn't you?). Then at your leisure you can boot DOS, LOAD the program from tape, and SAVE it on a diskette.

If you accidentally (or intentionally) hit the RESET key while the disk's red "IN USE" light is on, the information on your diskette may be clobbered. Problems are most likely to occur if this happens when you're putting information onto the diskette using a SAVE, BSAVE, or WRITE command. In the event that it is clobbered, you probably won't be able to recover your programs from the diskette. If nothing else works, you can re-initialize the diskette and use it again, but INITializing destroys all the files on the diskette.

If your IN USE light stays on for several minutes but you don't hear the usual disk sounds, your system may be "hung". Pressing RESET may be the only way to turn off the light so you can restart the system.

A diskette can be partially clobbered, so that it will not boot. However, in such a circumstance, you can sometimes boot another diskette, then LOAD programs from the partially clobbered diskette and save them on an un-damaged diskette. Or you may find that the UPDATE program (see Chapter 5) may help you save the day and your programs at the same time.

CHAPTER 3
EXERCISING OPTIONS

DRIVE, SLOT AND VOLUME OPTIONS

Most DOS commands allow you to specify a number of options, such as which disk drive you are using, which slot contains the disk controller for that drive, and a "volume number" for the disk.

The disk drive option allows you to operate with more than one drive. Each controller has the ability to control either one or two disk drives. Normally, instructions refer to drive 1. This is the default drive selection: if you don't specify a drive, drive 1 will be used. If you wish to specify drive 2, you use the notation D2 separated from the file name or other disk options by a comma. For example, to initialize a diskette in drive 2, you could use the instruction

```
INIT HELLO, D2
```

After drive 2 has been specified, all further disk commands refer to drive 2 until drive 1 is again specified. Drive 2 is now the default drive. After the above INITialization, the command

```
CATALOG
```

will list the files stored on the diskette in drive 2. To specify drive 1, you use the notation D1 separated from the file name by a comma. For example,

```
CATALOG, D1
```

will show you the contents of the diskette in drive 1, and change the default drive number back to 1.

If more than two drives are in use, then additional controllers are required. These are placed in different slots than the first controller (which is customarily in slot number 6). You can specify slot n (where n is a digit from 1 to 7) with the notation Sn separated from the file name or other disk options by a comma. For example, to initialize a diskette in drive 1 attached to a controller in slot 5, you would use the instruction

```
INIT HELLO, S5, D1
```

The file name must come first, but order of the options is not important.

The default slot number is the one you used when booting the DOS. Once a different slot number has been specified, it becomes the default slot number until it is explicitly changed.

After using a DOS command with a Slot parameter naming a slot that doesn't contain a disk controller, you get an

```
I/O ERROR
```

message, and all appears to be fine. But DOS now thinks the default slot number is the bad slot number, and that the disk that isn't connected to that slot is still running. Even if the next DOS command specifies the right slot, it waits in limbo forever for the non-existent disk to stop running the last command. If you have no program in memory that you care to save, simply re-boot DOS. To recover with your program intact, do this:

1) Reset the default slot by typing

```
CATALOG, Ss
```

where s is the correct slot number.

2) When the system hangs, press the RESET key.

3) Type
   3DØG
   and all should be fine again.

DOS must be booted from a diskette in Drive 1 not Drive 2.

The volume number option can be used to protect diskettes from being accidentally written over. For example, suppose your have a diskette-based inventory system, where each month's records is on a different diskette with a unique volume number. Then when you go to enter information for the month of January, you must be sure to specify the correct volume number. Otherwise, the information won't be written to the diskette and you'll get a
VOLUME MISMATCH
message.

A "volume number" may be assigned to a diskette when it is INITialized, using the notation Vn separated from the file name or other disk options by a comma. For instance, to initialize a diskette using the name "START UP" for the greeting program (the program that is run each time the diskette is booted), where the diskette is in drive 2 of a controller in slot 5, assigning the diskette a volume number of 128, you would use the command
INIT START UP, D2, S5, V128

The volume number of a diskette may not be changed without re-INITializing the diskette.

The drive number, slot number and volume number options may appear in any order. The above command is equivalent to
INIT START UP, V128, S5, D2
and to
INIT START UP, S5, V128, D2
and so on.

The volume number of a diskette must be an integer from 1 through 254. If no volume number is specified with INIT, a default volume number of 254 is assigned to the diskette.

The command
INIT HELLO, V0
does not give any message, but assigns the diskette the default volume number 254.

All DOS commands can specify the volume number, if you wish DOS to check that the volume number on the diskette agrees with the V option. If you do not specify any volume number, or if you specify volume zero, or if you type "V" without a number, DOS will ignore the diskette's volume number. If you accidentally specify an incorrect volume number, the system will reject it with the message
VOLUME MISMATCH

Volume mismatch errors cannot occur when you ask to see the CATALOG. In case you wish to know the volume number of a diskette, it is given at the head of the CATALOG listing.

Additional discussion of options is found where each command is introduced. Also, the information is concisely summarized for each command in the Command Summary Appendix and on the Quick Reference Card accompanying this manual. The following section explains how to interpret these concise summaries.

SYNTAX

Syntax refers to the structure of a computer command, the order and correct form of the command's various parts. A simple notation is used to describe the syntax of each DOS command. Items in brackets ([and]) are optional; optional parts of a DOS command may be specified in any order. CAPITAL letters and commas must be typed as shown; lower case letters stand for items you must supply. In specifying the syntax for DOS commands,

f	stands for a file name
d	stands for drive number -- either 1 or 2.
s	stands for slot number -- 1 through 7.
v	stands for volume number -- 1 through 254, usually. A diskette's volume number may not be Ø. Specifying a volume number of Ø in a disk command is a "wild card" that tells DOS to ignore the volume number on the diskette.

Additional abbreviations used in this manual are summarized at the start of the Command Summary Appendix.

Any numerical constant (the drive number, volume number, etc.) in a DOS command can be expressed in hexadecimal notation by preceding the hex digits with a dollar sign. If you don't know what hexadecimal notation is, ignore the preceding statement -- you need't understand hex notation to understand this manual.

INIT

The syntax for the INIT command is
INIT f [,Vv] [,Ss] [,Dd]
where the brackets indicate options which may or may not be included. The example
INIT HELLO, V17, D2
can be interpreted as follows.

The command name "INIT" is in upper case, and must be typed exactly as shown. The lower case "f", for file name is replaced by the legitimate file name "HELLO". Next the optional volume number is indicated: first comes a comma, then the upper case "V". The "v" for volume number was arbitrarily replaced by 17 for this example. The brackets around ",Ss" indicate that specifying the slot number is optional for the INIT command:

in this example it's omitted so DOS will use the default slot number. The drive option is included: the comma and upper case "D" must be as shown; the lower case "d" is replaced by 2 in this example.

For details on using INIT, see "INITializing New Diskettes" in Chapter 2.

LOAD, RUN AND SAVE

LOADing, RUNning and SAVEing programs on the disk is similar to the corresponding operations using the cassette (except that programs are referenced by file name). Everything goes at least ten times faster, and you never need to press buttons to play, record or rewind. It is all automatic. There are many additional abilities that the disk brings as well, such as the catalog of programs and the automatic running of programs without user intervention. Saving data (on text files -- see Chapter 6) is also very easy.

It's a good idea to hang on to your cassette tape system for trading programs and as back-up storage for vital programs and data (although experience shows that disk storage is even more reliable than cassette storage of programs and data).

If you have a program in BASIC, and you wish to call it HENRY, then the command
SAVE HENRY
will save it on the diskette. If you have more than one drive, HENRY normally would be saved on the drive from which you booted DOS (the default drive, unless you specified a different drive after booting). You can specify drive number, volume number and slot number as with the INIT command. For example, to SAVE a file called AGATHA on drive 1 of the controller in slot 2, where the volume number of the diskette is 214, you could use the command
SAVE AGATHA, D1, S2, V214
As before, the three options can be put in any order. If you had omitted the volume number option, AGATHA would have been saved just the same, blues her, but DOS would not have checked that the diskette was volume 214.

Program names are file names, and must follow the file name rules: they may be up to 30 characters long, and must start with a letter. They may include any characters you can type except commas or control characters. Here are some valid names for files:
CHECKBOOK
THE QUALITY OF MERCY
HIRES34
NOW: HEAR THIS!

To LOAD a program named AGATHA, use the command
LOAD AGATHA
and the program of that name, if there is one in the catalog, will be loaded. To test if AGATHA is loaded, see if she can walk a straight line.

If you want AGATHA to RUN after she's LOADed (poor thing) you can, of course, use the commands
LOAD AGATHA

then
RUN
But there's a way to do it in just one step:
RUN AGATHA
is a DOS command that first LOADs the specified file, then RUNs it.

Here's the syntax for the SAVE and LOAD and RUN commands:
SAVE f [,Ss] [,Dd] [,Vv]
LOAD f [,Ss] [,Dd] [,Vv]
RUN f [,Ss] [,Dd] [,Vv]

Examples follow:
SAVE OUR HAPPY HOME, D1, S7
LOAD UP
RUN AMOK, S7

If, when you try to SAVE a program, you get a SYNTAX ERROR message, either you have made a typing error, or DOS isn't booted. First, try re-typing the command. If DOS was originally booted, use
IN#6
to try to recover. If DOS isn't booted -- DON'T BOOT IT. Booting DOS will erase any program in memory. First, save the program on tape, using the usual cassette
SAVE
command. Now boot DOS. Next, use the usual cassette
LOAD
command, to bring the program back into your APPLE II's memory from the tape. Now you will be able to SAVE it on disk.

If a diskette is bad (perhaps someone tried to staple it into a notebook), or if the diskette is not initialized, or if there is no diskette in the drive, or if the door is open, the message
I/O ERROR
(I/O stands for Input or Output) will appear when you try to SAVE or LOAD using DOS. Check all the items listed, and correct the problem. You don't need to re-boot DOS. Try again.

If you use the command
LOAD HENRY
and HENRY is not the name of a program on the diskette in the drive, then you get this message
FILE NOT FOUND
Look at the diskette's catalog to find the program's exact file name. All characters and spaces must be typed exactly as they appear in the file name shown in the catalog.

DELETE

To eliminate any file that you would rather not have on your diskette, the command
DELETE
can be used. The syntax is
DELETE f [,Ss] [,Dd] [,Vv]

For example, the command
DELETE EXCESS, V34, D2, S1
deletes a file named EXCESS from a diskette with volume number 34, which is placed in drive 2 of the controller in slot 1.

Sectors on a diskette are "set free" only when a file is DELETEd.

A SCENARIO: BOOT, SAVE, RUN, CATALOG AND DELETE

Suppose you're running Integer BASIC and the System Master diskette is in your disk drive. Here's a dialog as it might appear on the screen of your APPLE II. The parts you type are underlined, although they do not appear that way on the TV screen.

>PR#6

(the above will clear the screen and you'll see the following:)

```
[illegible]
                                    16 [illegible]
COPYRIGHT [illegible] APPLE COMPUTER IN[illegible]
[illegible]
[illegible]
*I [illegible] HELLO
*I 0[illegible] [illegible]
*  [illegible]
*  [illegible]
 I [illegible]
*I [illegible]
*  [illegible]
*A [illegible]
*A [illegible] LITTLE [illegible]
*  [illegible]
*  [illegible] TEXT
*A [illegible]
*A 010 [illegible]
*I [illegible] APPLE [illegible]
*A [illegible] INSTRUCTIONS
*A [illegible]
NEW
[illegible]
[illegible]
[illegible]
[illegible]
```

[At this point, you would insert the Slave diskette you INITialized earlier, since it is not write protected.]

```
>CATALOG
DISK VOLUME 254
 I 002 HELLO
 I 002 COUNT
>[illegible]E DEMO
```

```
]CATALOG
DISK VOLUME 254
 I 002 HELLO
 I 002 COUNT
 I 002 DEMO
]NEW
]RUN
*** NO END ERR
]RUN DEMO
]ARRAYDUCK
]DELETE DEMO
]LOAD HELLO
DISK VOLUME 254
 I 002 HELLO
 I 002 COUNT
```

MOVING BETWEEN LANGUAGES: FP AND INT

Suppose you've been using Integer BASIC, and you decide to write a program in APPLESOFT, or to use the computer as a calculator with floating point numbers (numbers with decimal points). To invoke APPLESOFT without clobbering DOS, type

```
FP
```

(that's all there is to it) and in a few seconds APPLESOFT will be up and running. The FP stands for "Floating Point", of course. (If for some reason Applesoft isn't available -- it's not in firmware or on the diskette that's in use -- then the message

```
LANGUAGE NOT AVAILABLE
```

will be displayed.) The syntax for the command is

```
FP [,Ss] [,Dd]
```

where the optional Slot and Drive parameters allow to to specify the drive containing Applesoft on a diskette.

If you've been using APPLESOFT and DOS, you can type

```
INT
```

(for "Integer BASIC") to return to Integer BASIC with DOS intact. The suntax for this command is simply

```
INT
```

without any parameters. You'll generate a

```
SYNTAX ERROR
```

message if you try to use the D or S parameters with INT.

If you type

```
INT
```

while in Integer BASIC, you will lose any program in memory. Similarly, if you type

```
FP
```

while in Applesoft, you'll lose any program in memory.

When you switch from Integer BASIC to Applesoft or vice versa, you'll lose any program you happen to have in memory.

In addition to moving back and forth between the Apple's BASICs, you may wish to enter the Monitor and be able to use DOS commands. To do so from either Applesoft or Integer BASIC, type

CALL -151

and you should get the Monitor prompt character, * . To return to whichever BASIC you started from with your program and DOS intact, type

3D0G

From the Monitor, you may also type

INT

to return to Integer BASIC, or

FP

to return to Applesoft; in either case, DOS will still work but any program in memory will have disappeared.

If you get a

PROGRAM TOO LARGE

message when trying to execute an

FP

command, type

INT

first, to reset the system. Then type

FP

Even though your diskette contains the Integer BASIC program named APPLESOFT, do not type

RUN APPLESOFT

If you do, Applesoft will seem to be running fine until you press RESET, say, and try to re-enter Applesoft. Then, since the DOS thinks you are in Integer BASIC (because APPLESOFT was an Integer BASIC program), you will be in trouble.

To move the APPLESOFT program from one diskette to another, simply

LOAD APPLESOFT

from whatever diskette it's on, then place the diskette you wish to contain Applesoft in the drive and type

SAVE APPLESOFT

USE OF DOS FROM WITHIN A PROGRAM

Very often it's useful to be able to execute a DOS command from within a BASIC program. For example, you may wish your greeting program on a disk to print out the contents of the disk by doing a CATALOG command. Many DOS commands can be executed from inside a BASIC program. This is done by PRINTing a string that consists of a CTRL-D followed by the command.

Here is an Applesoft program that, if used as a greeting program, will cause the information in the PRINT statements in lines 20 and 30 to appear on the screen, followed by a list of files in the CATALOG.

```
5  REM   GREETING PROGRAM
10 D$ =  CHR$ (4): REM   CHR$ 4
     IS CTRL D
20  PRINT "SLAVE DISKETTE CREATED
     ON 32K SYSTEM"
30  PRINT "BY ANY DOS ON A 48OR
     GT 1000"
40  PRINT D$ "CATALOG"
50  END
```

The recommended way to do this in Applesoft is illustrated in the above program. First the string D$, consisting only of a CTRL-D, is created using the CHR$ function in the first line of the program. Later it can be used as in line 40
40 PRINT D$; "CATALOG:"
Note the semi-colon after the D$ and the quotation marks around the DOS command. The semi-colon is optional in Applesoft PRINT statements, so if a program has many DOS commands in PRINT statements, you may find it saves typing time and memory space to simply omit them, and use the form
40 PRINT D$"CATALOG"

In Applesoft, you can use the CHR$ function to specify CTRL-D
10 D$=CHR$(4): REM CTRL-D
But you need to recall that the ASCII code for CTRL-D is 4, so a REMark may be useful. (The CHR$ function is not available in Integer BASIC.)

In either Integer BASIC or Applesoft you may define CTRL-D by typing the characters
D$="
then typing the letter D while holding down the CTRL key, and then typing the quotation mark, ". Note that the CTRL-D does not print on your TV screen. The final command will appear as
D$=""
Since control characters do not print, it's often a good idea to follow with a REMark to remind you of what actually is in the string. Here's the above program written in Integer BASIC:

```
10 D$="" REM  THERE IS AN INVISIBL
   E CTRL-D BETWEEN THE QUOTES
20 PRINT "SLAVE DISKETTE CREATED ON
    32K SYSTEM"
30 PRINT "BY ANY DOS ON A 48OR GT
   1000"
40 PRINT D$ "CATALOG"
50 END
```

Only one DOS command may be used in a PRINT statement. The PRINT statement must begin with the CTRL-D and end with the DOS command.

Using the right-arrow to copy a BASIC statement containing an invisible control character will erase the control character.

In DOS commands executed by a program, the D$ must be preceded by a RETURN or it will be ignored. RUNning this program

```
 5  REM   TESTCATALOG PROGRAM
10 D$ =  "  REM   THERE IS AN INV
     ISIBLE CTRL-D BETWEEN THE QU
     OTES
20  PRINT "TEST";
30  PRINT D$;"CATALOG"
40  END
```

will cause
TESTCATALOG
to be displayed, since the semi-colon suppresses the RETURN at the end of the PRINT command in line 20. To correct this, and cause the DOS command CATALOG to be executed when the program is RUN, just delete the semi-colon (;) from the end of line 20.

These DOS commands should only be used within programs in a PRINT statement beginning with a CTRL-D:

OPEN
APPEND
READ
WRITE
POSITION

These DOS commands may be used in immediate-execution mode, and also from within a program using a PRINT command with CTRL-D:

CATALOG	BSAVE
SAVE	BLOAD
LOAD	BRUN
RUN	EXEC
DELETE	CLOSE
RENAME	CHAIN
LOCK and UNLOCK	PR#
MON and NOMON	IN#

The DOS command MAXFILES may be used as described above in an Applesoft program, but it must be used in a special way from an Integer BASIC program, as discussed in the section about the EXEC command in Chapter 7.

The DOS command INIT should be used only in immediate-execution mode (dire consequences may result if you ignore this admonition).

CHAPTER 4
PLAYING SAFE

Two ways of protecting you and/or your diskettes against disaster have already been mentioned. Chapter 3 mentions using the Volume option to ensure that you place information on the desired diskette. The use of control characters in file names can also be used as a way of protecting yourself (see Chapter 2, "What's in a Name?" and also Appendix F, "File Names"). If what appears in the CATALOG as

MY BANK ACCOUNT

in fact has your initials placed as control characters at some point(s) in the name, then it's unlikely that anyone else can access the file.

This chapter mentions a variety of ways of protecting you and your diskettes against various undesirable events. You'll probably find one or more of the techniques useful at one time or another. First, consider making a special purpose "turnkey" system.

CREATING A TURNKEY SYSTEM

Suppose a doctor wants to do the office accounting on an APPLE II. Ideally, the office staff should be able to simply turn on the APPLE II, type

{RESET} 6{CTRL}P {RETURN}

and immediately be in the midst of the doctor's accounting program. Since the accounting program would (hopefully) communicate with the user in ordinary English, the staff wouldn't need to know BASIC or anything else about the APPLE II. The computer would become an accounting system, its internal characteristics unimportant since all the staff needs to know is how to use the accounting program.

This is the essence of a "turnkey" system: from the user's point of view the computer is a device that does only a particular task, and getting the system started is as simple as turning a key in a lock. In this case, the "key" is simply turning on the Apple's power switch and pressing five keys on the keyboard. It does not require computer expertise to be able to do that.

You can use the diskette's "greeting program," named when you INITialized the diskette, to turn your APPLE II into a turnkey system. Let's say that you wanted the computer to run the COLOR DEMO program (provided on the System Master diskette) every time you booted Disk II. Here's how:

1) INITialize a blank diskette, as described in Chapter 2.

2) place the System Master diskette in your drive and type
   RUN COLOR DEMO
   Once you're satisfied that the program RUNs correctly, type
   {CTRL}C
   to stop the program and return to BASIC.

3) Put your newly INITialized diskette into your drive. We'll assume that you called your "greeting" program HELLO when you INITialized the diskette.

4) The program COLOR DEMO is now in memory. When you type
   SAVE HELLO
   DOS will erase your original greeting program named

HELLO and save the COLOR DEMO program under the HELLO file name. The COLOR DEMO program is now the greeting program on your diskette.

To check that all works as expected, boot the disk. You should get the same program that you used in step 2).

You've just created a turnkey system: whenever that diskette is booted, it will automatically LOAD the COLOR DEMO program and RUN it.

LOCK AND UNLOCK

Sometimes you'll want to prevent a particular program from accidentally being erased from a diskette: the LOCK command will do this for you.

Example:
LOCK NESS, D2

The CATALOG of this diskette's contents will now show an asterisk (*) next to the entry for NESS.

If you decide you no longer wish to keep the file LOCKed, the UNLOCK command will (surprise!) unlock the file.

Example:
UNLOCK NESS

The syntax for the commands is
```
  LOCK f [,Ss] [,Dd] [,Vv]
UNLOCK f [,Ss] [,Dd] [,Vv]
```
The interpretation of the notation is discussed in the Syntax section of Chapter 3.

If you try to DELETE or RENAME a file that's LOCKed you'll receive the message
FILE LOCKED
You'll also see this message if you try to SAVE a file using the name of a LOCKed file (if the file you're trying to SAVE is in the same language as the LOCKed file).

If you try to SAVE a file using the name of a LOCKed file in a different language, then you'll receive the message
FILE TYPE MISMATCH
Try again, using a different file name.

VERIFY

Occasionally information may not be recorded correctly on a diskette. This may happen if the diskette is scratched or dirty, for example. The VERIFY command reports a file which may be damaged or written incorrectly.

The syntax is the usual one for DOS commands:
VERIFY f [,Ss] [,Dd] [,Vv]
Examples of the way to use the command follow:
VERIFY SAM
VERIFY FINANCE-8,D2,V22

VERIFY checks to see that information in the specified file is self-consistent. If it is, you see no message: the prompt character for the language you're using is simply printed:
> for Integer BASIC
] for Applesoft
* for the Monitor.
However, VERIFY doesn't check to see whether or not a program is clobbered. If you SAVEd a program that was messed up somehow, it will still be messed up on the diskette, and it will still VERIFY.

If the VERIFY command finds an error, the
I/O ERROR
message is displayed.

If you try to VERIFY a file that isn't on the disk, the message
FILE NOT FOUND
is presented.

You can use VERIFY from Integer BASIC, Applesoft, or the Monitor. From these languages you may VERIFY any type of file, including text files (see Chapters 6, 7 and 8) and machine language programs (see Chapter 9).

WRITE-PROTECTING A DISK

The LOCK command allows you to protect a particular file. But sometimes you will want to be sure that all files on a certain diskette are not accidentally written over, and thus lost. To "write-protect" a diskette, you merely need to cover up the squarish write-protect cutout in the side of the disk. Stick-on adhesive labels are supplied for this purpose when you purchase diskettes but, in a pinch, any piece of sturdy tape will do. Note that the System Master diskette does not have a write-protect cutout: it is permanently write-protected.

If you decide you want to re-use a write-protected diskette, simply remove the label (often called a "tab") that covers the write-protect cutout.

Some programs cannot be used with a write-protected diskette. An example of such a program is ANIMALS, one of the demonstration programs of the System Master disk. Put your System Master in your drive, and boot DOS if you need to. Now type

```
LOAD ANIMALS
```

which will put the program into memory. Now type

```
RUN
```

and the message

```
WRITE PROTECTED
STOPPED AT 1040
```

will be displayed. ANIMALS won't RUN on a write-protected diskette because it saves information on the diskette each time you play the game. When you RUN the program, the diskette in the drive must not be write-protected, else the information can't be written on the diskette.

Now ANIMALS is in memory, but you can't RUN it with the System Master diskette. Put an initialized diskette, one that is not write-protected, in the drive. Next type

```
RUN
```

and now you can play ANIMALS, a game that will "remember" what you "teach" it by saving the information on the diskette. When you're through playing, type

```
SAVE ANIMALS
```

so that you'll have the game on a diskette that's not write-protected.

If you type

```
CATALOG
```

you should see that you have not only a copy of ANIMALS on the diskette, but also a new file called ANIMALSFILE that was created by the program ANIMALS.

PROTECTING YOURSELF AGAINST DISASTER

Floppy disks are sturdy and reliable compared to some other ways of storing computer programs -- for example, on the backs of old envelopes. But it's still possible to lose or destroy all information on a diskette. A diskette may get scratched or damaged by heat; it may get lost, or a dog may chew it; someone may decide to use it as a frisbee at the beach; if a diskette isn't write-protected, it may accidentally get written over. And a diskette will eventually wear out -- a lifetime of 40 working hours is about average.

** Moral **

Keep more than one copy of a program around if you don't want to lose it. In computerese, "back up" any valuable program.

If you are in the midst of writing or modifying a program, one way to back up the program is to keep copies of earlier versions. Then if the current version is lost you can fall back to the next-most-recent version, and hopefully not lose too much programming time. One good way to do this is

to end each file name with a number which changes from version to version. For example, suppose you start to write a program called FINANCE. The first time you save the program, call it FINANCE-1. Next time you work on the program, save it under the name FINANCE-2; the third time, it becomes FINANCE-3, and so on. You'll wind up with a whole collection of FINANCE programs, with the largest version number representing the most recent version of the program.

It's a good idea to SAVE a developing program periodically (with a new version number). If you do this every 15 or 20 minutes, an unexpected power failure or other disaster will not erase all your work. You can, of course, immediately continue working after SAVEing the current state of the program -- just be sure to assign a new version number for the next SAVE. If the diskette starts filling up, DELETE some of the earlier versions. But it's a good idea to keep several versions around, in case something calamitous happens to the current version. Or you may just happen to want an earlier version -- not all revisions are improvements.

The phrase "backing up" is also used to describe keeping multiple copies of programs on separate diskettes. There are two approaches to backing up in this fashion. The first method works with only one drive: simply SAVE the program on one diskette, remove that diskette from the drive, insert another diskette and SAVE the program again.

The second approach involves duplicating all the information from one diskette onto a second diskette; this works only if you have more than one disk drive. This approach is discussed in the next section.

USING THE COPY PROGRAM

Those of you with one disk drive will have to LOAD, then SAVE, programs one by one onto the diskettes you use for back-up copies of programs. Those of you with two drives can use the COPY program, on the System Master diskette, to copy the entire contents of your current programming diskette onto your back-up diskette. The COPY program requires that both disk drives must be connected to the same disk controller card, to prevent overstraining the power supply.

In the COPY program, the diskette from which copying is done is called the "original." The entire contents of the source diskette will be copied onto a "duplicate" diskette. The "duplicate" diskette does not have to be INITialized before being copied onto. In fact, any previous information that was on the "duplicate" diskette will be erased.

Before copying the "original" diskette, it's a good idea to write-protect it. Then you can't accidentally erase its contents, even if you put it into the wrong drive.

As default values, the program assumes that the "original" diskette will be placed in the currently selected drive (the drive from which you ran COPY), connected to the controller card in the currently selected slot. To use the default slot or drive number for the "original" diskette, just press the RETURN key when the program expects you to type a number. If either default is wrong for the "original" diskette in your system, you must type the correct number when it's requested by the program.

When you have specified the slot and drive numbers for the "original" diskette, the COPY program tells you where the "duplicate" diskette must be placed. It will go in the remaining drive controlled from the same slot which you specified for the "original".

Here's an example of using the COPY program with the default slot and drive numbers. It assumes your two disk drives are connected to a disk controller card in slot #6.

1) Place the System Master diskette in drive 1. Type
   RUN COPY
   and after the usual whirring you should see

```
        APPLE DISKETTE DUPLICATION PROGRAM

              PLEASE INSERT ORIGINAL


      ORIGINAL  SLOT        DEFAULT = 6
```

2) Remove the System Master diskette from drive 1 and then insert the original diskette, from which you wish to copy, into drive 1 (did you remember to write-protect your original?).

3) Now press the RETURN key to indicate you want to use the default slot number, slot #6 in our example, for the original diskette. Next you'll see

```
                DRIVE       DEFAULT   1
```

   and again press the RETURN key to indicate you want the default drive number, drive 1 in our example, for the original diskette.

4) Now you will see

```
        APPLE DISKETTE DUPLICATION PROGRAM

              PLEASE INSERT ORIGINAL

      ORIGINAL  SLOT        6
                DRIVE       1

      DUPLICATE SLOT        6
                DRIVE       2


            ORIGINAL IN SLOT 6
                     DRIVE 1
           DUPLICATE IN SLOT 6
                     DRIVE 2

            INSERT DUPLICATE THEN
      --- PRESS 'RETURN' KEY TO BEGIN COPY ---
```

Insert the "duplicate" diskette, onto which you wish to place the back-up copy, into the drive specified for it: drive 2, in this example. To stop the program at any response, use the traditional CTRL-C

5) Finally, to begin copying, press the RETURN key.

If the message

```
****** UNABLE TO WRITE ******
```

occurs when you try to copy a diskette, there is no diskette in the "duplicate" disk drive. The message

```
***** DUPLICATE DISK IS PROTECTED *****
```

indicates a problem with the "duplicate" diskette or the drive containing it. If the diskette has a tab over its write-protect notch, no information can be put onto the diskette until the tab is removed.

In other situations, DOS will report an
I/O ERROR
if a diskette is inserted improperly or if the drive door is left open, but the COPY program will tell you (incorrectly) that the diskette is write-protected.

If you do not have a second disk drive connected, you will be given the incorrect message that the "duplicate" is write-protected.

The message

Indicates a problem with the "original" diskette or the drive containing it. Perhaps the diskette has been clobbered, or perhaps there's no diskette in the drive.

When the program asks

```
DO YOU WISH TO MAKE ANOTHER COPY?
```

answer with Y for YES, or N for NO, and press the RETURN key. Do not type more than one character before pressing RETURN.

If you try to use a printer with the COPY program, you'll find weird displays both on the TV screen and on the printer.

CHAPTER 5
MORE "HOUSEKEEPING" INFORMATION

DEBUGGING: MON AND NOMON

The process of trying to get a program to run the way you want it to is called "debugging;" program errors are often referred to as "bugs". All disk commands and all information sent between the computer and the disk are normally not displayed on the screen. But when you're debugging, monitoring this information can help you track down problems.

The MON command allows you to MONitor a variety of information. To turn various parts of the display off again, use the NOMON (NO MONitor) command.

Three different parameters that may be used in these commands:
C stands for Commands to the disk (such as OPEN, READ, etc)
I stands for Input from the disk (when READing a file)
O stands for Output to the disk (when WRITEing a file).
These parameters are used only with the NOMON and MON commands. Usually NOMON C,I,O is in effect: no monitoring is taking place.

The syntax for the commands is

```
  MON [C] [,I] [,O]
NOMON [C] [,I] [,O]
```

At least one of the three parameters must be present with the NOMON and MON commands, else the command will be ignored. The parameters may appear in any order and, as usual, must be separated by commas.

There are 7 different ways in which the MON command may be used:

command	what it monitors
MON C	Commands to the disk
MON I	Input from the disk
MON O	Output to the disk
MON I,O	Input from and Output to the disk
MON C,I	Commands to and Input from the disk
MON C,O	Commands to and Output to the disk
MON C,I,O	Commands to, Input from, and Output to the disk

To illustrate how the command works, a sample program called TEST MON is included on the System Master diskette. To try out the program, place the System Master in your drive and type

```
LOAD TEST MON
```

then SAVE the program on a diskette that's not write-protected. Now RUN the program. A list of options will be presented to you. Each time you select an option the program will put material into a disk file and also retrieve material from a disk file. (Lines 100 through 180 create a text file SAMPLE containing 3 numbers and 2 strings; lines 200 through 270 retrieve the file SAMPLE.) Try out all the options: notice what kind of information is displayed for each possible combination of parameters.

*** NOTE ***

A MON command remains in effect until a
NOMON, INT, or FP (firmware only) command is encountered
or
until you boot the system
or
do a restart (3D0G).

A neat trick: you can issue a MON command and later cancel it without affecting the screen format -- even the NOMON does not show on the screen. Suppose you execute a MON command, say

```
MON C, I, O
```

To cancel the command without having it print on the screen, include

```
PRINT D$; "NOMON C,I,O"; CHR$(37): CALL -868
```

where D$, as usual, contains CTRL-D.

MAXFILES

DOS allows up to 16 files to be active (in use) at one time. DOS deals with several types of files in addition to the BASIC program files discussed so far. See Chapter 6 for a discussion of sequential text files, Chapter 8 for random-access text files, and Chapter 9 for the DOS commands used with binary (machine language) files.

The MAXFILES command specifies how many active files are permitted. When you boot DOS, the command

```
MAXFILES 3
```

is executed, which sets up the default condition: a maximum of 3 files may be active simultaneously until another MAXFILES command is executed.

The command's syntax is

```
MAXFILES n
```

where n must be an integer from 1 to 16. Specifying a value outside this range will cause a SYNTAX ERROR message from either Applesoft or Integer BASIC; from the Monitor, a beep is the only indication that you've done something wrong.

For each file specified, MAXFILES sets aside 595 bytes of memory space called a file buffer. This additional memory space for each active file is used to help adjust for the fact that memory speed is far faster than disk access speed, which involves mechanical motion -- the disk head has to search the diskette. So in the name of efficiency, a file buffer is used to "buffer" information going to and from a diskette.

If you retrieve information from a diskette, DOS brings in 256 characters at a time and puts them in the "input" part of the file buffer, then delivers to you whatever subset of those 256 characters your program requested. If you are sending information to a diskette, characters are stored in the "output" part of the file buffer until 256 characters have accumulated, then they're whipped to the diskette all at once.

Suppose you have MAXFILES 1 and one file is active. An attempt to perform a DOS command (such as CATALOG) will cause the message

```
NO BUFFERS AVAILABLE
```

to be displayed.

When the system is booted, the number of active files (n) defaults to 3, so 1785 bytes of memory are reserved for 3 file buffers. Under most circumstances, you won't need more than 3 active files. If more files are required, type

```
MAXFILES n
```

(where n is the number of needed files) in immediate execution mode before loading and running a program.

In immediate execution mode, increasing MAXFILES erases Integer BASIC programs and messes up Applesoft strings, since HIMEM: is moved down without moving the program or strings. To avoid this problem, reset MAXFILES before loading and running a program.

If MAXFILES is used within a program, it changes memory pointers, and a GOTO, GOSUB, or other instruction can get lost. If you must change MAXFILES from within an Applesoft program, make the MAXFILES command the first statement in the program, before any string variables are declared. For example,

```
1Ø  PRINT CHR$(4); "MAXFILES 5"
2Ø  REM  REST OF PROGRAM GOES HERE
```

To use MAXFILES from within an Integer BASIC program, you need to create an EXEC file, as discussed on page 78.

TRACE

The Applesoft TRACE command is a useful debugging tool. But when TRACE is in effect, DOS commands inside Applesoft programs don't work because TRACE prints the line number with no RETURN before the DOS command. There's a partial solution to the problem. You can insert a RETURN (CHR$(13)) into the D$ string

```
1Ø D$ = CHR$(13) + CHR$(4)
```

and then most DOS commands will work properly even if TRACE is in effect.

If TRACE is in effect when DOS (with the D$ fix, above) is WRITEing, all the TRACEd line numbers will be printed into your text file along with the characters <u>you</u> wished to print.

USING THE UPDATE PROGRAM

As discussed in Chapter 2, INIT is used to create slave diskettes. In this section you'll learn how to create master diskettes. The distinction between a slave and a master is not readily apparent: both come charmingly attired in the latest in black plastic (no, <u>not</u> leather) garb. It's up to you to devise your greeting program and diskette label to remind yourself which is slave and which is master.

The System Master diskette contains a program called UPDATE 3.2 that can run on an Apple II with at least 16K of memory. The UPDATE program does the following for you:

* "Updates" a previously INITialized diskette to DOS version 3.2 without affecting program or data files that are already stored on that diskette.

* Converts a slave diskette (whose DOS is memory-size dependent) into a master diskette (whose DOS is self-relocating so that memory is used efficiently on any size system).

* Gives the updated diskette a new greeting program name, the name DOS will attempt to RUN each time the diskette is booted.

The UPDATE 3.2 program must be used with a diskette that has already been INITialized. It will not work with a diskette that is write-protected.

Here's an example of how to UPDATE the diskette INITialized in Chapter 2 (the one with the ONE TO TEN program on it) to convert the slave diskette created by INIT into a master diskette. For convenience, that diskette will be referred to as diskette ONE in the discussion that follows.

Before using UPDATE 3.2, do the following:

1) Insert the diskette you wish to update -- diskette ONE for this example -- into the disk drive and RUN the diskette's greeting program -- named HELLO on diskette ONE. The message displayed by a greeting program should include the version of DOS used to initialize the diskette, and its status as slave or master.

2) Change the appropriate lines of the greeting program to display the new information, "MASTER DISKETTE UPDATED TO DOS VERSION 3.2". Then SAVE this new version of the greeting program. If the diskette's outside label requires a similar change, make that change now.

3) Note the name of the greeting program. If you wish the updated diskette to RUN this same program each time it is booted, just as it did before updating, you will give this greeting program name to the UPDATE 3.2 program, later on. If you've always wished that your greeting program had some other name than its present one, RENAME the greeting program now. Later, you will give the new name to the UPDATE 3.2 program.

To use UPDATE 3.2, do the following:

4) Put the System Master diskette in the drive, boot DOS, and from either BASIC type
BRUN UPDATE 3.2

5) You should see the message

```
DOS 3.2 MASTER  UPDATE UTILITY
COPYRIGHT 1979 BY APPLE COMPUTER INC
ALL RIGHTS RESERVED

(NOW LOADING DOS IMAGE)
```

6) You'll then be told to type the greeting program name to be used by the updated diskette:

```
PLEASE INPUT THE "GREETING" PROGRAM'S
FILE NAME
```

We'll assume that when you SAVEd the revised greeting program on diskette ONE (step 3, above), you used the name HELLO. So type
HELLO
unless you wish the diskette to RUN some other program name each time the diskette is booted. When you press the RETURN key to enter the greeting program name, you'll see this message:

```
REMEMBER THAT "UPDATE" DOES NOT CREATE
THE "GREETING" PROGRAM OR PLACE IT IN
THE DISK DIRECTORY

  THIS IS THE FILE NAME THAT WILL BE
PLACED WITHIN THE IMAGE

      HELLO

  PLACE THE DISKETTE TO BE "UPDATED" IN
THE DISK DRIVE

  PRESS (RETURN) WHEN READY

NOTE - IF YOU WANT A DIFFERENT FILE NAME,
PRESS (ESC)
```

4) Follow the instructions. Remove the System Master diskette from the disk drive and replace it with the diskette you wish to update -- diskette ONE, in this case. Finally, press the RETURN key to begin updating; the program will inform you when the process is complete.

5) After using the UPDATE 3.2 program, always re-boot DOS before doing any other work.

*** Note ***

The greeting program name that you give to the UPDATE 3.2 program is not placed in the diskette's CATALOG. It just tells the diskette's DOS which program name to RUN each time the diskette is booted. You must make sure that the diskette's CATALOG actually contains a program bearing the same name you give to the UPDATE 3.2 program.

If you forget to do so (by skipping step 3, above), you'll see the message
FILE NOT FOUND
each time you boot the disk using this diskette.

*** Reminder ***

You must remember the name of the greeting program for each diskette. You can make this pretty simple by using the same greeting program name on all of your diskettes.

CHAPTER 6
USING SEQUENTIAL FILES

TEXT FILES: AN INTRODUCTION

Sometimes you'll want to use the disk to store information that is not a program. You may, for example, wish to keep copies of correspondence, a list of words used in a word-guessing game, intermediate results of a calculation, or a mailing list. A text file, sometimes called a data file will allow you to do this and more. The letter T marks text files in the CATALOG directory.

Text files are created and retrieved using DOS commands in an Integer BASIC or Applesoft program. A text file may be created using a program written in one language and retrieved from a diskette using a program written in another language.

Most sample programs in this manual are in Applesoft. If you wish to convert the programs to Integer BASIC, recall that in Integer BASIC you can't make string arrays and you must DIMension string variables. In an Integer BASIC command such as
INPUT A$, B$, C$
only RETURNs (not commas) may separate the three responses. This manual does not tell you how to make each program run in Integer BASIC; see the Appendix M of the Applesoft II BASIC Programming Reference Manual for details of converting between languages. For some hints about changing the BASIC in which a program runs, after the program has been written, see page 76 of this DOS manual.

The DOS commands LOAD and RUN (also BLOAD and BRUN) may not be used with a text file. An attempt to do so will cause the message
FILE TYPE MISMATCH
to appear. LOAD and RUN expect a BASIC program file (and BLOAD and BRUN expect a Binary machine-language file), not a text file. Instead, you must write programs that send data to a text file and retrieve data from a text file, using the DOS commands discussed in this chapter:
OPEN
CLOSE
READ
WRITE
APPEND
POSITION
EXEC

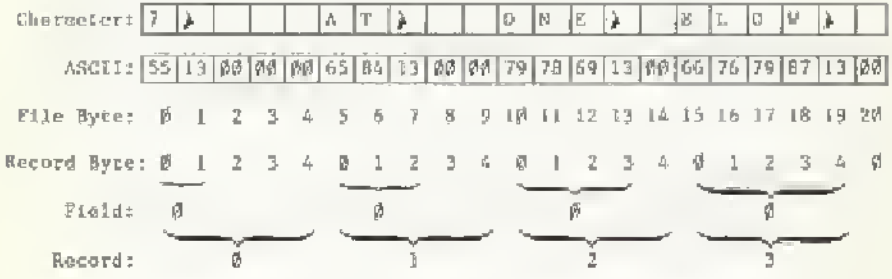
The commands OPEN, READ, WRITE, APPEND and POSITION cannot be used in immediate-execution mode. If you try to do so, you'll receive the message
NOT DIRECT COMMAND
These commands must be used in deferred-execution mode, that is, from within a program. The commands CLOSE and EXEC may be used in immediate-execution mode.

In addition to the commands listed above, the DOS commands
LOCK and UNLOCK
DELETE
RENAME
MON and NOMON
VERIFY
CATALOG
work with text files in the same way they work with program files.

There are two different types of text files: sequential text files and random-access text files. Both types of text files store strings of ASCII codes to represent the data, but in different formats. Diagrams of the two text-file types are shown below (the character ♪ represents the RETURN character, sent automatically at the end of most PRINT statements).

"PICTURE" OF A SEQUENTIAL TEXT FILE

"PICTURE" OF A RANDOM-ACCESS TEXT FILE
(Example: Record Length 5, One Field per Record)



The terms "field" and "record" will be discussed in Chapters 6, 7 and 8. The commands OPEN, CLOSE, READ, WRITE and POSITION are used with both types of files, but in somewhat different ways. Sequential text files are simpler to use and understand, in some respects, so we will discuss the use and structure of sequential text files first. The use of random-access text files is described in Chapter 8. More detailed and technical information about all types of files can be found in Appendix C.

SEQUENTIAL TEXT FILES: SOME EXAMPLES

Suppose you want to make a file containing a list of words to be used in a word-guessing game. Here are two pairs of programs that deal with such a file. The first program in each pair creates a text file on the diskette. The second program in each pair retrieves the data stored in the text file from the diskette.

This program creates a text file named WORDS1, containing the words APPLE, BANANA, CATALOG, DORMANT, EAGLE, FRUIT, GOOSE, HAT and ICICLE.

```
10  REM  MAKE WORDS1
20 D$ = "": REM  CTRL-I
30  PRINT D$;"OPEN WORDS1"
40  PRINT D$;"WRITE WORDS1"
50  PRINT "APPLE"
60  PRINT "BANANA"
70  PRINT "CATALOG"
80  PRINT "DOMINANT"
90  PRINT "EAGLE"
100  PRINT "FRUIT"
110  PRINT "GOOSE"
120  PRINT "HAT"
130  PRINT "ICICLE"
140  PRINT D$;"CLOSE WORDS1"
150  END
```

Line 30 OPENs the file, using the normal format for sending a DOS command from within a BASIC program. OPEN places a text file named WORDS1 in the CATALOG (if it was not there previously).

Line 40's WRITE command causes subsequent output from PRINT statements to be sent to the named text file instead of to the TV screen. So in this program, each PRINT statement in lines 50 through 130 will send the word inside the quotation marks to the text file WORDS1, and not to the TV screen.

Line 140 CLOSEs the file, and ends the file-writing process.

If the program is RUN and you're not in MONitor mode you won't see anything: usually DOS commands and disk input and output are not displayed. But if, as explained in Chapter 5, you type

```
MON C, I, O
```

(or simply

```
MON C, O
```

where no input from the disk is involved) and then you RUN the above program you'll see the following:

```
OPEN WORDS1
WRITE WORDS1
APPLE
BANANA
CATALOG
DOMINANT
EAGLE
FRUIT
GOOSE
HAT
ICICLE
CLOSE WORDS1
```

At this point you'll have a file called WORDS1 on your diskette. WORDS1 will be marked with a "T" in the CATALOG to indicate that it's a text file. The file consists of items of data (in this case, words) separated by RETURNs. A RETURN character is automatically sent at the end of every PRINT statement which does not end with a comma or a semicolon. Note that in this sense each RETURN is a *character* rather than an *action* — in particular, it is the character with ASCII code 13.

Each item of data, ending with its RETURN character, is called a *field*. A field is stored in the text file as a series of characters represented by their ASCII codes. The last character in each field must be the RETURN character, ASCII code 13.

WORDS1 is called a *sequential text file* because each field is stored beginning immediately after the RETURN character of the preceding field. When stored on the diskette, fields may be of different lengths: the word APPLE takes 6 bytes (one for each letter plus one for the RETURN character), BANANA takes 7 bytes, and so on. A sequential text file is stored on the diskette as one long, continuous series of ASCII-coded characters, a chain of fields with no gaps left between them.

Once WORDS1 is on the diskette, the question immediately arises, "How can I retrieve it?" The following Applesoft program will retrieve WORDS1:

```
10  REM   RETRIEVE WORDS1
20 D$ = CHR$ (4)  REM   CTRL-D
30  PRINT D$;"OPEN WORDS1"
40  PRINT D$;"READ WORDS1"
50  FOR I = 1 TO 9
60  INPUT A$(I)
70  NEXT I
80  PRINT D$;"CLOSE WORDS1"
90  END
```

Line 30 OPENs the file; line 40 tells DOS that all subsequent INPUT or GET statements will refer to the named diskette file instead of the Apple's keyboard. It is as if the disk were typing responses, instead of you. An INPUT command always causes *one complete field*, ending with its RETURN character, to be "typed in" to the Apple. If another INPUT command follows, it will cause the *next* field to be read in, and so on. So lines 50 through 70 cause DOS to start at the beginning of WORDS1 and retrieve 9 fields which are placed in the array A$(1), A$(2), A$(3), A$(4), ... A$(9). Line 80 politely CLOSEs the file.

If MON C, I, O is not in effect when the above program is RUN, you will see nothing on your screen. But if MON C, I, O (or just MON C, I) *is* in effect, you'll see

```
OPEN WORDS1
READ WORDS1
APPLE
BANANA
```

```
?CATALOG
?DORMANT
?EAGLE
?FRUIT
?GOOSE
?HAT
?ICICLE
CLOSE WORDS1
```

A question mark (?) is displayed before each INPUT from the disk, just as it is before each normal keyboard INPUT.

To check that all worked as claimed, try typing
PRINT A$(2), A$(9), A$(4)
and you should see the words BANANA -- from A$(9) -- then ICICLE and finally DORMANT. This is a good way to check that information was read correctly.

If you modify the program MAKE WORDS1 to make different words, be sure to DELETE WORDS1 before re-RUNning MAKE WORDS1. If you don't, you may end up with a mixture of the old words and the new.

Here's how to create a sequential file called WORDS2 containing the same words as WORDS1, but with all nine words in one field. Each word is followed by a comma, so that an INPUT statement with multiple variables (9, in this case) can be used to retrieve the separate words.

```
10  REM    MAKE WORDS2
20 D$ = "": REM   CTRL-D
30  PRINT D$;"OPEN WORDS2"
40  PRINT D$;"WRITE WORDS2"
50  PRINT " APPLE, BANANA, CATALOG
      ,";
60  PRINT " DORMANT, EAGLE, FRUIT
      ,";
70  PRINT " GOOSE, HAT, ICICLE"
80  PRINT D$;"CLOSE WORDS2"
90  END
```

Note that the PRINT command in line 50 ends with a semi-colon. A semi-colon at the end of a PRINT command stops the automatic printing of a RETURN character after the last data character. Therefore the characters sent to the disk by the next PRINT command will appear in the same field with the characters sent by line 50's PRINT command. The PRINT command in line 60 also ends with a semi-colon, so the field still does not have its end-marking RETURN character. Line 70's PRINT command ends without a semi-colon or comma, allowing the automatic final RETURN character to be sent at last. This ends the field, which now contains all the characters PRINTed by lines 50, 60 and 70.

Commas in a disk-less PRINT command usually send characters to defined tab-fields on the screen. However, commas do <u>not</u> serve this same formatting function in PRINT commands used when WRITEing to the disk: these commas are treated as if they were semi-colons. In PRINTing to the disk, items separated by commas will be concatenated, with no intervening spaces inserted. A comma at the end of a PRINT command has the same effect as a semi-colon: no automatic final RETURN character is sent.

When the program MAKE WORDS2 is RUN with MON C, I, O in effect, you'll see

```
OPEN WORDS2
WRITE WORDS2
 APPLE, BANANA  ATALOG, DORMANT, EAGLE,
 FRUIT, GOOSE, HAT, ICICLE
CLOSE WORDS2
```

This Applesoft program retrieves WORDS2:

```
10  REM     RETRIEVE WORDS2
20 D$ = "" : REM   CTRL-D
30  PRINT D$;"OPEN WORDS2"
40  PRINT D$;"READ WORDS2"
50  INPUT A1$,A2$,A3$,A4$,A5$,A6$
     ,A7$,A8$,A9$
80  PRINT D$;"CLOSE WORDS2"
90  END
```

When the above program is RUN with MON C, I, O in effect, you'll see

```
OPEN WORDS2
READ WORDS2
? APPLE, BANANA, CATALOG, DORMANT, EAGLE
, FRUIT, GOOSE, HAT, ICICLE
CLOSE WORDS2
```

In Integer BASIC, commas can separate multiple INPUT responses for <u>numeric</u> variables, but not for <u>string</u> variables. Only RETURN characters can separate multiple responses when INPUT is used with multiple string variables. In Integer BASIC, therefore, the program RETRIEVE WORDS2 will assign the entire field (9 words, 8 commas and 6 spaces) to the variable A1$. Then you will get the END OF DATA message when there is no field to assign A2$.

In Applesoft BASIC, you can also use the GET command to retrieve data from a text file, character by character. This has the advantage that you can define <u>any</u> character as marking the end-of-word, for instance. The following Applesoft program also retrieves the text file WORDS2.

In line 10, the CLEAR command sets all variables (including I and all A$(I)'s) to zero. Line 20 uses Applesoft's alternate way of setting D$ to CTRL-D (4 is the ASCII code for CTRL-D). This method avoids the invisible (and un-copyable) control character.

```
10  CLEAR   REM  GET WORDS2
20  D$ = CHR$ (4):  REM  CTRL-D
30  R$ = CHR$ (13):  REM  RETURN
40  T$ = CHR$ (1):  REM  CTRL-A
50  PRINT D$;"OPEN WORDS2"
60  PRINT D$;"READ WORDS2"
70  I = I + 1
80  GET B$
90  IF B$ = "," THEN  GOTO 70
100  IF B$ = R$ THEN  GOTO 130
110 A$(I) = A$(I) + B$
115  PRINT T$;A$(I)
120  GOTO 80
130  PRINT R$;D$;"CLOSE WORDS2"
140  END
```

Line 80 GETs one character at a time from the text file WORDS2, which was OPENed for READing in lines 50 and 60. If the new character is neither a comma nor a RETURN, line 110 adds the new character to the end of the string A$(I). Then line 120 sends the program back to line 80, to GET the next character. Thus, the program builds up the first word, character by character, in A$(1).

When a comma is found the first word is ended, so line 90 sends the program back to line 70 to increment I and start collecting a new word in A$(2). And so on. Finally, a RETURN character (R$) marks the end of the field, so line 100 sends the program on to line 130 to CLOSE the file and end the program. Note the use of CHR$(13), in line 30. You cannot directly type a RETURN character into a BASIC program line (a typed RETURN ends a program line), but CHR$(13) is a RETURN character in Applesoft.

When GET obtains characters from the disk, these characters are not displayed on the screen, even in MON C, I, O mode. Line 95 has been added to let you see the words as they are built up, character by character.

After an Applesoft GET command takes its response from a diskette text file, the following problems arise:

1) With NOMON C,I,O the first character PRINTed after the GET will not appear on the screen.
2) With MON C,I,O the first character PRINTed after the GET will appear on the screen.
3) In either mode, if a DOS command is the first item PRINTed after the GET, the DOS command may not be executed because the necessary preceding RETURN is missing.

In the program GET WORDS2, the non-printing "throw-away" character CTRL-A (T$) was placed before the first desired PRINT character in line 115. This takes care of problems 1 and 2, above. To cure problem 3, the RETURN character (R$) was placed before the PRINTed DOS command in line 130, much as was done with TRACE (see page 44).

When this program is RUN with MON C, I, O in effect, you will see the following (but all displayed in one column, not three):

```
OPEN WORDS2
READ WORDS2

A                    D                    G
AP                   DO                   GO
APP                  DOR                  GOO
APPL                 DORM                 GOOS
APPLE                DORMA                GOOSE
                     DORMAN
B                    DORMANT              H
BA                                        HA
BAN                  E                    HAT
BANA                 EA
BANAN                ENG                  I
BANANA               EAGL                 IC
                     EAGLE                ICI
C                                         ICIC
CA                   F                    ICICL
CAT                  FR                   ICICLE
CATA                 FRU                  CLOSE WORDS2
CATAL                FRUI
CATALO               FRUIT
CATALOG
```

And lastly, here's an Applesoft program that creates a file WORDS3, with 2 words in the first field, 3 words in the second field, and 4 words in the third field.

```
10  REM  MAKE WORDS3
20 D$ =  CHR$ (4): REM CTRL-D
30  PRINT D$;"OPEN WORDS3"
40  PRINT D$;"WRITE WORDS3"
50  PRINT "APPLE,BANANA"
60  PRINT "CATALOG,DORMANT,EAGLE"

70  PRINT "FRUIT,GOOSE,HAT,ICICLE
    "
90  PRINT D$ "CLOSE WORDS3"
99  END
```

The first field will contain
APPLE,BANANA
and is 13 bytes long, one per character (commas must be counted too) plus one for the RETURN character. The second field,
CATALOG,DORMANT,EAGLE
is 22 bytes long; the third field,
FRUIT,GOOSE,HAT,ICICLE
is 23 bytes long.

When RUN with NOW C, 1, 0 in effect you'll see

```
OPEN WORDS3
WRITE WORDS3
APPLE,BANANA
CATALOG,DORMANT,EAGLE
FRUIT,GOOSE,HAT,ICICLE
CLOSE WORDS3
```

Here's a program to retrieve WORDS3:

```
10  REM  RETRIEVE WORDS3 A
20 D$ =  CHR$ (4)' REM  CTRL-D
30  PRINT D$;"OPEN WORDS3"
40  PRINT D$;"READ WORDS3"
50  INPUT R$,S$
60  INPUT T$,U$,V$
70  INPUT W$,X$,Y$,Z$
80  PRINT D$;"CLOSE WORDS3"
90  END
```

When RUN with NOW C, 1, 0 in effect, you'll see the following:

```
OPEN WORDS3
READ WORDS3
?APPLE,BANANA
?CATALOG,DORMANT EAGLE
?FRUIT,GOOSE,HAT,ICICLE
CLOSE WORDS3
```

The programs to READ the sequential text files WORDS1, WORDS2, WORDS3 were carefully designed to READ exactly the correct number of fields and the correct number of items per field. In general, a program to retrieve a text file must be designed around the specific file. If you make a mistake, the results can appear somewhat confusing. For instance, consider the following "wrong" program to retrieve the words in text file WORDS3.

```
10  REM  RETRIEVE WORDS3-B
20 D$ = "" REM  CTRL-D
30  PRINT D$;"OPEN WORDS3"
40  PRINT D$;"READ WORDS3"
50  INPUT R$,S$
60  INPUT T$,U$,V$
70  INPUT W$,X$,Y$
80  PRINT D$;"CLOSE WORDS3"
90  END
```

With NON C, I, O in effect, here's what you would see on RUNning the program.

```
OPEN WORDS3
READ WORDS3
?APPLE,BANANA
?CATALOG,DORMANT,EAGLE
?FRUIT,GOOSE,HAT ICICLE
?EXTRA IGNORED
CLOSE WORDS3
```

The INPUT command in line 70 caused the entire field containing
FRUIT,GOOSE,HAT,ICICLE
to be READ into the Apple. The first three words were assigned to the variables W$, X$ and Y$. But there is no variable corresponding to the fourth INPUT response, ICICLE, so the message
EXTRA IGNORED
is displayed, and execution continues.

Here is another "wrong" program to READ the text file WORDS3:

```
10  REM  RETRIEVE WORDS3 C.
20 D$ = "" REM  CTRL D
30  PRINT D$,"OPEN WORDS3"
40  PRINT D$,"READ WORDS3"
50  INPUT R7,S4
60  INPUT T8,U5,V6,W$
70  INPUT X$,Y1,Z2
80  PRINT D$,"CLOSE WORDS3"
90  END
```

And here is a NON C, I, O RUN of the program.

```
OPEN WORDS3
READ WORDS3
?APPLE,BANANA
?CATALOG,DORMANT,EAGLE
??FRUIT,GOOSE,HAT,ICICLE
?EXTRA IGNORED
?
END OF DATA
BREAK IN 70
```

This time, line 60 caused the field
CATALOG,DORMANT,EAGLE
to be READ into the Apple. The three words are assigned to the variables T$, U$ and V$. But line 60's INPUT command expected four responses, so it causes the next complete field to be READ into the Apple:
FRUIT,GOOSE,HAT,ICICLE
The first word, FRUIT, is assigned to line 60's last variable, W$. There are no more variables with this INPUT command, so the message
EXTRA IGNORED

is displayed, and execution continues. There are no more fields in the file, so line 70's INPUT command causes the
END OF DATA
message, and the program comes to a stop.

A somewhat more general pair of programs, MAKE TEXT and RETRIEVE TEXT are discussed in a later section. They illustrate how to make a program more adaptable to different text files.

OPEN-ING AND CLOSE-ING SEQUENTIAL FILES

Sequential text files should be used when information is to be retrieved in a linear fashion from the beginning to the end of the file, and when information does not require much updating or on-going revision. For example, a sequential file could be used to contain data for a word-guessing game, as in the preceding sample programs.

To create a sequential text file, the commands
OPEN
WRITE
PRINT
CLOSE
are used, in the order shown (though not necessarily right after each other). To retrieve a sequential text file, the commands
OPEN
READ
INPUT
CLOSE
are used, again in the order shown though not necessarily right after each other. Both procedures are illustrated in the preceding section.

A certain ritual is required before and after you create (WRITE) a sequential text file: before using the file you must OPEN it. When you're done, you must CLOSE it. The same is true when retrieving (READing) a sequential text file: OPEN before READing, and CLOSE the file when you're done.

Files that have been OPENed must be CLOSEd. Failure to CLOSE a file that was OPENed and written to by a WRITE command may result in loss of data.

The syntax for these commands is similar to other DOS commands.
[Note: OPEN and CLOSE are also used with random-access files -- see Chapter 8.]

```
OPEN # [,Ss] 1,Dd1 [,Vv]
CLOSE [#]

Examples: OPEN SESAME
          OPEN SHOP, D2, S7
          CLOSE
          CLOSE MOUTHED
          CLOSE WINDOW
```

OPEN sets aside workspace in the Apple for the file f (for those who know about such stuff, OPEN allocates a 595-byte file buffer to handle this file's input and output), and gets the system ready to read or write from the beginning of the file. OPEN also sets up the slot and drive numbers to be used by the subsequent WRITE (or READ) command.

The CLOSE command releases the workspace in the Apple (de-allocates the file buffer associated with the file f). If f is not specified, all OPEN files will be closed, with the exception of any file being used by the EXEC command. EXEC files are discussed later in Chapter 7. OPEN sometimes CLOSEs too: OPEN first checks to see if the named file is already OPEN; if so, it CLOSEs it before re-OPENing it.

Note that the CLOSE command has no Drive or Slot parameters. If you type
CLOSE MYFILE
then any file named MYFILE will be CLOSEd, regardless of the slot and drive number associated with the file. Similarly, the command
CLOSE
will CLOSE all files (except a file being EXECed) on all disk drives.
In various circumstances, you may wish to delete a file f that may or may not exist. This is especially important to avoid problems of overwriting an old file (unless you overwrite the entire old file, part of the old file will remain, attached to the end of your new file). Suppose a game creates and uses the file SCORES each time it is played, and you wish your program to delete any old file by that name at the start of each new game. The command
DELETE SCORES
will cause the error message
FILE NOT FOUND
if the file doesn't exist, and your program will halt. Here's a quick way to delete any file named SCORES and re-OPEN it for new data, whether or not that file already exists:

```
5  REM  SCORES DELETER
10 D$ = CHR$ (4): REM  D$ IS CTRL-D
15  PRINT D$;"OPEN SCORES"
20  PRINT D$;"DELETE SCORES"
25  PRINT D$;"OPEN SCORES"
30  REM
        REMAINDER OF PROGRAM
              HERE
```

WRITE-ING SEQUENTIAL FILES

Here is another program which creates a sequential text file. This Applesoft program creates a text file named SAMPLE which contains 3 strings and 10 numbers.

The file SAMPLE may or may not already exist each time the program is RUN: if it does exist, it should be DELETEd so as to remove old data from the file. If it does not exist and your program tries to DELETE it, you'll receive the message
FILE NOT FOUND

and the program will stop. Lines 20 and 30 take care of the problem. If SAMPLE already exists, line 20 OPENs it and line 30 DELETEs it. If SAMPLE does not exist, line 20 creates a file SAMPLE and line 30 DELETEs it. When line 40 is executed it creates a clean new file SAMPLE, so the problem of mixed files is avoided.

```
5  REM  MAKE SAMPLE
10 D$ =  CHR$ (4)  REM  CTRL-D
20  PRINT D$;"OPEN SAMPLE"
30  PRINT D$;"DELETE SAMPLE"
40  PRINT D$;"OPEN SAMPLE"
50  PRINT D$;"WRITE SAMPLE"
60  PRINT "HI HO"  PRINT "HI HO"
70  PRINT "OFF TO THE DISK WE GO"
80  FOR J = 1 TO 10
90  PRINT J
100  NEXT J
110  PRINT D$;"CLOSE SAMPLE"
120  END
```

Here's what you see on the screen when you RUN this program, if HOW C, I, O is in effect.

```
OPEN SAMPLE
DELETE SAMPLE
OPEN SAMPLE
WRITE SAMPLE
HI HO
HI HO
OFF TO THE DISK WE GO
1
2
3
4
5
6
7
8
9
10
CLOSE SAMPLE
```

Before you WRITE a file, it must be OPENed; CLOSE it (quietly, please) when you're done. Both the OPEN and WRITE commands must refer to the same file name.

Once a WRITE command is executed, any subsequent PRINT commands send all characters to the diskette, instead of the screen. A WRITE command is cancelled by the use of any DOS command in a PRINT statement. Even the "empty" DOS command (just CTRL-D) will do.

An INPUT command of the form
INPUT X$
also cancels a WRITE command, but only after storing as the last text file character the ? which the INPUT command normally displays on the screen. If the form
INPUT "WHAT'S YOUR NAME? "; X$
is used, the WRITE is canceled after the characters in the string are sent to the diskette.

An error message cancels a WRITE command, but only after the entire error message is stored as the last field in your text file.

The syntax for the WRITE command when used with sequential files is:
WRITE f
[Note: WRITE is also used with random-access files, see Chapter 8.]

Examples: WRITE LETTER
WRITE RIGHT

The sample program given at the beginning of this section is a simple illustration of the most basic (BASIC?) elements needed to create a text file. A slightly more general Applesoft program called MAKE TEXT is on the System Master diskette that came with your disk drive.

MAKE TEXT allows you to create a sequential text file containing up to 100 strings; each string may have at most 239 characters. Try it -- you'll like it (we hope). Place the System Master diskette in your drive and type
LOAD MAKE TEXT

A LISTing of the program should look like this:

```
5  REM   MAKE TEXT
10  DIM A$(100): I = 0
20 D$ = CHR$ (4): REM CHR$(4) IS
      CTRL-D
30  HOME : PRINT
40  PRINT "THIS PROGRAM LETS YOU
     WRITE TEXT FILES"
50  PRINT "YOU GET TO TYPE ONE ST
     RING AT A TIME"
60  PRINT "A STRING MAY HAVE UP T
     O 239 CHARACTERS"
70  PRINT : I = I + 1
80  PRINT "(TO QUIT, PRESS RETURN
     KEY FIRST)"
90  PRINT "TYPE STRING #";I;" ";
```

(Continued on next page)

(Continued from previous page)

```
100  INPUT "";A$(I)
110  IF A$(I) <  > "" GOTO 70  REM
       FIRST KEY PRESSED WAS NOT
       RETURN KEY
120  PRINT
130  INPUT "WHAT FILE NAME? ";N$
140  PRINT D$;"OPEN ";N$
150  PRINT D$;"DELETE ";N$
160  PRINT D$;"OPEN ";N$
170  PRINT D$;"WRITE ";N$
180  PRINT I - 1
190  FOR J = 1 TO I - 1
200  PRINT A$(J)
210  NEXT J
220  PRINT D$;"CLOSE ";N$
```

Once the program is LOADed, SAVE it on a diskette that's not write-protected. (This step is necessary because this program, like the ANIMAL5 program discussed in Chapter 4, creates a new file.)

Is MAKE TEXT still in your Apple? And a non-write-protected disk in the drive? If so, type

```
NON C, I, O
```

so you can see the commands sent to and from the disk. Then type RUN and you should see the following message:

```
THIS PROGRAM LETS YOU WRITE TEXT FILES

YOU GET TO TYPE ONE STRING AT A TIME

A STRING MAY HAVE UP TO 239 CHARACTERS

(TO QUIT, PRESS RETURN KEY FIRST)
TYPE STRING #1.
```

Type in as many strings as you like (up to 100 may be entered). Warning: the program uses INPUT, so don't type commas or colons into your strings. When you wish to quit, just press the RETURN key instead of typing a string. You'll be asked

```
WHAT FILE NAME?
```

Choose a name for your text file, press the RETURN key, and as your strings are sent to the disk you'll see them printed on the screen. First will appear the disk commands

```
OPEN f
DELETE f
OPEN f
WRITE f
```

(where the f is replaced by the file name you chose). They'll be followed by a number -- the number of strings you entered into the file. (This

number will be used by a program discussed in the next section that retrieves your file). Next you'll see your strings. Finally you'll see the message
CLOSE £

Here's a sample RUN of the MAKE TEXT program:

```
      THIS PROGRAM LETS YOU WRITE TEXT FILES

      YOU GET TO TYPE ONE STRING AT A TIME

      A STRING MAY HAVE UP TO 255 CHARACTERS


      (TO QUIT, PRESS RETURN KEY FIRST)
      TYPE STRING #1  HERE'S STRING 1

      (TO QUIT, PRESS RETURN KEY FIRST)
      TYPE STRING #2  AND MY SECOND STRING

      (TO QUIT, PRESS RETURN KEY FIRST)
      TYPE STRING #3  OH BE GO

      (TO QUIT, PRESS RETURN KEY FIRST)
      TYPE STRING #4  ENOUGH ALREADY!

      (TO QUIT, PRESS RETURN KEY FIRST)
      TYPE STRING #5

      WHAT FILE NAME? TEST
      OPEN TEST
      DELETE TEST
      OPEN TEST
      WRITE TEST
      4
      HERE'S STRING 1
      AND MY SECOND STRING
      OH BE GO
      ENOUGH ALREADY!
      CLOSE TEST
```

If you OPEN a text file that already exists and then WRITE to it (without first DELETEing the file and re-OPENing it), then you will overwrite at least a portion of the file. Unless you overwrite at least as many characters as existed in the old file, the result is that the new file contents will be a mix of the data PRINTed to the file on the two occasions. First will appear the new characters you PRINTed to the file

this time, and then will follow any portion of the old file you did not overwrite. To clear all characters from the old file, OPEN and DELETE the old file before you OPEN it anew. (In the program MAKE TEXT, lines 140 and 150 take care of "cleaning out" any previous text file by the same name.) To keep programs from overwriting a file, LOCK the file.

READ-ING SEQUENTIAL FILES

The DOS command READ allows you to retrieve a text file. Once a READ is executed, any subsequent INPUT statements (or GETs in Applesoft) refer to the specified file instead of the Apple's keyboard. This Applesoft program retrieves the text file SAMPLE created by the program listed at the beginning of the preceding section. READ, like WRITE, must be preceded by OPENing the file to be used. The file must be CLOSEd as well.

```
5  REM  RETRIEVE SAMPLE
10 D$ =  CHR$ (4): REM   CHR$(4)
     IS CTRL D
20  PRINT D$;"OPEN SAMPLE"
30  PRINT D$;"READ SAMPLE"
40  INPUT A$,B$,C$
50  FOR I = 1 TO 10
60  INPUT W
70  NEXT I
80  PRINT D$;"CLOSE SAMPLE"
```

An OPEN must precede a READ, and an INPUT (or, in Applesoft, a GET) must follow a READ. The OPEN and READ must refer to the same file name. If you RUN the program with MON C, I, O in effect you'll see this:

```
OPEN SAMPLE
READ SAMPLE
??W1 HO
??W1 HO
??OFF TO THE DISK WE GO
?1
?2
?3
?4
?5
?6
?7
?8
?9
?10
CLOSE SAMPLE
```

The program was written explicitly with the SAMPLE file in mind: it assumes that the text file contains 3 strings, (A$, B$, and C$ in line 40) and 10 integers (W in line 60). Two question marks are printed when B$ and C$ are INPUT because RETURNs separated the INPUT's multiple responses.

A READ command is cancelled by the use of any DOS command in a PRINT statement. The "empty" DOS command (just CTRL-D) will do just fine. Use of the PR# or IN# commands also cancels a READ.

The syntax for the READ command is the same as for WRITE:
READ ‡
[Note: READ is also used with random-access files, see Chapter 8.]

Examples: READ LETTER
READ CAREFULLY

Stopping a READ in Applesoft using CTRL-C will generate a string of REENTERs. To avoid this, press the RESET key to stop the program.

An Applesoft program that receives text files created by the MAKE TEXT program is on the System Master diskette. Place the System Master diskette in your drive and type
LOAD RETRIEVE TEXT
then SAVE the program on the same diskette you used for MAKE TEXT. (The program is really a companion piece to MAKE TEXT, and it's simply more convenient to have them on the same diskette.)

A LISTing of the program should appear as follows:

```
5  REM   RETRIEVE TEXT
10 D$    CHR$ (4)  REM  CTRL-D
20  HOME   PRINT "THIS PROGRAM RE
     TRIEVES TEXT FILES"
30  PRINT "CREATED BY THE  MAKE T
     EXT  PROGRAM       "
40  PRINT "NON C,I,O IS IN EFFECT
      "
50  PRINT
60  INPUT "NAME OF TEXT FILE? ";Z
     $
70  PRINT D$;"NON C,I,O"
80  PRINT
90  PRINT D$;"OPEN ";Z$
100  PRINT D$;"READ ";Z$
110  INPUT I
120  DIM A$(I)
130  FOR J = 1 TO I
140  INPUT A$(J)
150  NEXT J
160  PRINT D$;"CLOSE ";Z$
170  PRINT D$;"NONON C,I,O"
```

Now type
RUN
and you should see the message

```
THIS PROGRAM RETRIEVES TEXT FILES
CREATED BY THE 'MAKE TEXT' PROGRAM
MUNI C.1.0 IS IN EFFECT.

NAME OF TEXT FILE?
```

Type in the name of the text file you created using the MAKE TEXT program, press the RETURN key, and you should be off and running (oops -- rather, READing).

Here's what you'll see if the file TEST, used as a sample at the end of the last section, is retrieved using the RETRIEVE TEXT program:

```
THIS PROGRAM RETRIEVES TEXT FILES
CREATED BY THE 'MAKE TEXT' PROGRAM
MUNI C.1.0 IS IN EFFECT

NAME OF TEXT FILE? TEST

OPEN TEST
READ TEST
?4
'HERE'S STRING 1
?AND MY SECOND STRING
?ON WE GO
?ENOUGH ALREADY"
CLOSE TEST
MUNI C.1.0
```

MORE ON SEQUENTIAL FILES: APPEND AND POSITION

The DOS commands APPEND and POSITION, respectively, allow you to add text to the end of a sequential text file, and to access information from any specified field within a text file.

APPEND allows you to add data to the end of a sequential text file. This is particularly useful if you wish to extend the information in a sequential text file, as in the ANIMALS program discussed in Chapter 4 could have. The command OPEN, you will recall, always sets the position-in-the-file pointer to byte 0, the first character in the file. The command APPEND performs an OPEN for you on a file that already exists, then sets the position-in-the-file pointer to one byte beyond the last character in the file.

The following program builds a file called TESTER that contains the two strings "TEST 0" and "TEST 1":

```
5  REM   MAKE TESTER
10 D$ =  CHR$ (4): REM  CTRL-D
20  PRINT D$;"OPEN TESTER"
30  PRINT D$;"DELETE TESTER"
40  PRINT D$;"OPEN TESTER"
50  PRINT D$;"WRITE TESTER"
60  PRINT "TEST 0"
70  PRINT "TEST 1"
80  PRINT D$;"CLOSE TESTER"
```

The following program APPENDs the strings "TEST 2", "TEST 3" and "TEST 4" to the file TESTER:

```
5  REM   APPEND TESTER
10 D$ =  CHR$ (4)  REM  CTRL-D
20  PRINT D$;"APPEND TESTER"
30  PRINT D$;"WRITE  TESTER"
40  PRINT "TEST 2"
50  PRINT "TEST 3"
60  PRINT "TEST 4"
70  PRINT D$;"CLOSE TESTER"
```

The following program displays the file TESTER:

```
10  REM   RETRIEVE TESTER
20 D$ =  CHR$ (4): REM  CTRL-D
30  PRINT D$;"OPEN TESTER"
40  PRINT D$;"READ TESTER"
50  FOR I = 1 TO 5
60  INPUT A$
70  NEXT I
80  PRINT D$;"CLOSE TESTER"
```

APPEND must be followed by WRITE (attempting to READ will just cause the END OF DATA message). The syntax for the APPEND command is doubtless familiar if you've been reading straight through this manual:
APPEND f [,Ss] [,Dd] [,Vv]

APPEND, even though it is used only for WRITEing into a text file, does not cause the
FILE LOCKED
message if the file is locked. That message is given only if you attempt to actually WRITE to the file.
The DOS command POSITION allows you to WRITE or READ information beginning in any given field of a sequential text file. The syntax for the POSITION command is
POSITION f [,Rp]
where Rp is the Relative-field position. This command specifies that DOS's position-in-the-file pointer will be moved forward (only) to the p-th field ahead of the current pointer position. If p=0, the following READ or WRITE begins in the current field. If p=1, the following READ or WRITE skips the current field and begins in the next field. If p=2, the

following READ or WRITE skips two fields including the current field, before beginning to READ or WRITE. And so on. If your file does not contain any field corresponding to the relative-field position specified by the POSITION command, the message
END OF DATA
will be displayed, and program execution will stop.

POSITION with the Rp parameter specifies a relative field position; p fields ahead of the current field. POSITION must refer to a file that you have already OPENed. OPEN automatically sets the position-in-the-file pointer back to the beginning of the first field. Thus, if POSITION is used immediately after OPEN, the relative-field position also corresponds to the actual, or absolute, field position. In no other case is this true.

Like any other DOS command, POSITION cancels a READ or a WRITE. Therefore POSITION must be used before the associated READ or WRITE.

POSITION actually scans the contents of the file, byte by byte, looking for the Rp-th RETURN character. If, during this process, it encounters an "empty" (value ∅) byte, the message
END OF DATA
is presented immediately. It is not necessary to actually INPUT or GET any such null character.

Here is a program that uses POSITION to retrieve various fields from the TESTER file, created earlier by the MAKE TESTER and APPEND TESTER programs:

```
10  REM   POSITION TESTER
20 D$ =  CHR$ (4): REM  CTRL-D
30  PRINT D$;"OPEN TESTER"
40  PRINT D$;"POSITION TESTER,R2"
50  PRINT D$;"READ TESTER"
60  INPUT A$
70  PRINT D$;"POSITION TESTER,R1"
80  PRINT D$;"READ TESTER"
90  INPUT B$
100  PRINT D$;"OPEN TESTER"
110  PRINT D$;"POSITION TESTER,R3
     "
120  PRINT D$;"READ TESTER"
130  INPUT C$
140  INPUT E$
150  PRINT D$;"CLOSE TESTER"
```

If you RUN this program with MON C, I, O in effect, you will see:

```
OPEN TESTER
POSITION TESTER,R2
READ TESTER
?TEST 2
POSITION TESTER,R1
```

```
READ TESTER
?TEST 4
OPEN TESTER
POSITION TESTFP,B3
READ TESTER
?TEST 3
?TEST 4
CLOSE TESTER
```

Are you surprised at the results? Remember that the current field is relative-field position number Ø. Also, remember that each INPUT causes one field to be READ into the Apple, and advances the position-in-the-file pointer to the beginning of the next field.

BYTE-ING OFF MORE

Note: the following section is not for beginners, and sequential files can be used perfectly well without a knowledge of the parameters discussed here.

The DOS commands WRITE and READ can be used with a Byte parameter to WRITE or READ information starting from any place in a text file -- if you know where that place is. The trick involves knowing at exactly which byte in the file you want to start (each byte contains one character's ASCII code). To do this, you must know exactly how you have stored information into the file. You must count all RETURNs, commas, spaces and other characters in the file when figuring out where to begin. The problem is even more difficult for WRITE, because you must also know where to end.

The B parameter is an actual or absolute position in the file unless R is specified. If R is given, the B parameter is the actual position within the specified field.

The command
WRITE THISMONTH, B27
sets the position-in-the-file pointer to the twenty-eighth byte of the file named THISMONTH (the first byte is number Ø). Characters sent to the disk by a subsequent PRINT command will replace an equal number of characters that already existed in the file, beginning with the character in the 28th byte.

This over-writing is not confined to the current field. If you PRINT fewer than the number of characters remaining in the current field, you will create two new fields: the field you just PRINTed, followed by the tail-end of the field you were over-writing. If you PRINT more than the number of characters remaining in the current field, you will over-write some of the characters at the start of the next field: the current field will then be longer, and the next field shorter than before.

It is also possible to WRITE into bytes that are beyond the last byte of an existing sequential text file. An attempt to READ the intervening un-written bytes will cause the

END OF DATA

message to be displayed, and your program will stop. See the discussion of READ with the B parameter, for information on accessing sequential text file fields that are not next to each other.

The syntax for this command is

WRITE f [,Bb]

where the B parameter specifies the file byte at which characters sent by the next PRINT command will begin replacing file characters. The default value of b is 0, the first byte in a file. Byte b is an actual, or absolute, position within the file. The B parameter may specify a position either before or after the current position-in-the-file pointer. [Note: this command is also used with random-access files. See Chapter 8.]

Similarly, the command

READ LASTMONTH, B32

sets the position-in-the-file pointer to the thirty-third byte of the file named LASTMONTH (again, the first byte is number 0). A subsequent INPUT command will cause all characters in the next field (i.e. up to the next RETURN character), beginning with the character whose ASCII code is stored in the file's 33rd byte, to be READ into the Apple. If the 33rd byte does not contain the first character of a field, only the remaining characters in that field will be READ.

Syntax for this command is

READ f [,Bb]

where the B parameter specifies the file byte where the next INPUT or GET command will begin reading characters. The default value of b is 0, the first byte in a file. Byte b is an actual, or absolute, position within the file. The B parameter may specify a position either before or after the current position-in-the-file pointer. [Note: this command is also used with random-access files, see Chapter 8.]

The following program sets the position-in-the-file pointer to byte 14 (the fifteenth byte) in file TESTER, which was created earlier by the program MAKE TESTER. Then it WRITEs the string "APPLE COMPUTER". Note the familiar sequence: OPEN, then WRITE and PRINT, and finally CLOSE.

```
5  REM   BYTE WRITER
10 D$ = CHR$ (4): REM  CTRL-D
20  PRINT D$;"OPEN TESTER"
30  PRINT D$;"WRITE TESTER,B14"
40  PRINT "APPLE COMPUTER"
50  PRINT D$;"CLOSE TESTER"
```

With NOW C, I, O in effect, RUN RETRIEVE TESTER to see how the previous program has changed the file TESTER. As you can see, the field containing APPLE COMPUTER has completely over-written the fields that contained TEST

3 and TEST 4, as well as the first character of the field that contained TEST 5. As there are now only four fields in all, the END OF DATA message was displayed after the fifth INPUT command.

The following program sets a pointer to byte 18 in the file TESTER, just modified by the preceding program. Then this program READs to the next RETURN in the file. Again the familiar format: OPEN is followed by READ, next some INPUT statements (or, in Applesoft, GETs may be used) and finally the file is CLOSEd.

```
5  REM   BYTE READER
10 D$ = CHR$ (4)  REM  CTRL-D
20  PRINT D$;"OPEN TESTER"
30  PRINT D$;"READ TESTER,B18"
40  INPUT A$
50  PRINT D$;"CLOSE TESTER"
```

Try to <u>predict</u> what you will see, before you RUN this program.

CHAPTER 7
AUTO APPLE

To better understand the contents of this chapter, it is suggested that you first read Chapter 6, on sequential text files.

CONTROLLING THE APPLE VIA A TEXT FILE: EXEC

The DOS command EXEC is similar to RUN, except that the disk file used by an EXEC command is a text file that contains commands or program lines, including BASIC statements, as if they were typed at the keyboard.

To initiate a demonstration of some EXEC command abilities,
LOAD EXEC DEMO
from your System Master diskette and then SAVE it on a diskette that's not write-protected. Leave the un-write-protected diskette in the drive, since the program WRITEs a text file.

Next RUN the program. You should see the message

```
                << EXEC DEMO >>
    THIS PROGRAM CREATES A SEQUENTIAL TEXT
    FILE NAMED "DO'ER" CONTAINING SEVERAL
    STRINGS, EACH A LEGAL APPLE IT COMMAND.
    WHEN YOU TYPE
    EXEC DO'ER
    THEN THE COMMANDS IN FILE DO ER TAKE
    CONTROL OF YOUR COMPUTER  EACH COMMAND
    WILL BE EXECUTED JUST AS IF IT HAD BEEN
    TYPED AT THE KEYBOARD  THE DOS MANUAL
    DESCRIBES THE PROGRAM IN MORE DETAIL
            << HAPPY EXECUTING >>
    PRESS THE SPACE BAR TO MAKE THIS
    PROGRAM CREATE THE FILE DO'ER
    IF YOU WISH TO STOP THIS PROGRAM NOW,
    YOU MAY PRESS THE ESC KEY
```

Press the Apple's space bar, and after a brief pause you should see the disk drive's IN USE light come on as the program writes the DO'ER file onto the diskette. Now type
EXEC DO'ER
press the RETURN key. Your Apple will begin a solo performance based on the script in the DO'ER file.

Here's a brief summary of the major things DO'ER does:
- First DO'ER issues a MON C, I, O command, so you'll be able to see what happens.
- Second, a three-line program is written and saved on diskette under the name NEW PROGRAM!! The program is then LISTed.
- Now a FOR-NEXT loop is executed to take up some time, so you get a chance to look at the screen before the activity continues.

Next DO'ER uses the INT command to enter Integer BASIC, LOADs the program COLOR DEMOS, and LISTs it.

At this point, DO'ER uses CALL -155 to enter the Monitor and executes some machine-language instructions before using the FP command to enter Applesoft.

From Applesoft a MON C,I,O command is executed, then NEW PROGRAM!! is RUN, modified, LISTed (again a FOR loop allows you to take a look at the screen) and SAVEd using the name EVEN MORE RECENT PROGRAM!!

Lastly, the program NEW PROGRAM!! is DELETEd and the CATALOG (including the new addition EVEN MORE RECENT PROGRAM!!) is displayed.

And you won't even have to lay a finger on the keyboard (unless your CATALOG has more than 10 entries, in which case you need to press the space bar to see the balance of the CATALOG entries).

CREATING AN EXEC FILE

Here's a step by step example to illustrate how to create an EXEC file named DOIT that contains the following commands:

```
LIST 20, 50
RUN AWAY
CATALOG
```

First create and SAVE an Applesoft program called AWAY to use in the above demonstration:

```
5  REM  AWAY
10  PRINT "AWAY AWAY WITH RUN BY
    GUM"
```

Next write and SAVE the following program, called MAKE EXEC, that will create a text file called DOIT. When you later EXEC DOIT, the commands your MAKE EXEC program has PRINTed into the DOIT text file will tell Apple to RUN the AWAY program for you. Notice that the commands which are PRINTed into the DOIT file, for later EXECing, are <u>not</u> preceded by a CTRL-D.

```
5  REM   MAKE EXEC
10 D$ =  CHR$ (4)  REM CHR$(4) IS
       CTRL-D
20  PRINT D$;"OPEN DOIT"
30  PRINT D$;"WRITE DOIT"
40  PRINT "LIST 20,50"
50  PRINT "RUN AWAY"
60  PRINT "CATALOG"
70  PRINT D$;"CLOSE DOIT"
```

After you have MAKE EXEC and AWAY both SAVEd on a diskette, type the command

```
RUN MAKE EXEC
```

to create a sequential text file named DOIT.

Type the command
EXEC DOIT
to cause the commands in the file DOIT to be executed one by one, just as if they'd been typed in from the keyboard. Again, notice that the commands now being EXECuted were not preceded by a CTRL-D in the program MAKE EXEC. First lines 20 through 50 from the program currently in memory (probably the program MAKE EXEC) are LISTed. Then the program named "AWAY" is RUN, and finally the CATALOG on the diskette is displayed.

CAPTURING PROGRAMS IN A TEXT FILE

Here's a far more useful example of using the EXEC command: it allows you to capture program listings as text files. Such a program can be used for

* translating Integer BASIC programs into Applesoft
* renumbering parts of programs and EXECing them anywhere into another program
* inserting favorite subroutines into programs from a subroutine file on the diskette by EXECing the file
* "appending" one program to another
* repairing programs that have become partially unreadable (you can capture the good portion in a text file, re-boot, then EXEC the program portion back into memory)

The line numbers 2270 and 5130, following the LIST command in line 6 of the CAPTURE program, should be replaced by the line numbers of the program in memory that you wish to capture. The name of the sequential text file containing the listing is LISTING.

```
1  REM  CAPTURE
2 D$ =  CHR$ (4): REM  CTRL-D
3  PRINT D$;"OPEN LISTING"
4  PRINT D$;"WRITE LISTING"
5  POKE 33,30
6  LIST 2270,5130
7  PRINT D$;"CLOSE LISTING"
8  TEXT : END
```

We made the line numbers of this program very close together, so that you can add these lines to a program already in memory, or anywhere within your program that you have eight free line numbers. You could just as easily put all the lines of CAPTURE above the highest numbered line in your program.

CAPTURE creates a text file containing commands that are preceded by line numbers. When you EXEC that text file, the numbered commands will not be executed. Instead, just as if you had typed those lines in from the keyboard, the lines are stored as a program in Apple's memory. Once captured in a text file, a program can be modified and then EXECed back into Apple's memory. Unlike LOAD or RUN, EXEC does not delete a program that is already in memory. Using CAPTURE, you can capture a program in a text file from one language, then EXEC the program back into another language (of course, the program may not run without some changes — there's somewhat different syntax for Integer BASIC and Applesoft). You

can also use EXEC this way to add new lines to an existing program in memory. In fact, you can save a listing of CAPTURE in a text file named LIST SAVER, say, and then EXEC LIST SAVER any time you wanted to add the CAPTURE program to a program in memory.

CONVERTING MACHINE-LANGUAGE ROUTINES TO BASIC

Here's another useful program that will take a machine-language routine and convert it into a BASIC program portion which POKEs the machine-language routine into memory. The program portion can be used as part of either an Applesoft or an Integer BASIC program, to put the machine-language routine into memory each time the BASIC program is run.

```
5  REM  CODE-POKES WRITER
10 D$ = "" : REM  CTRL-D
15  PRINT D$;"OPEN CODE-POKES"
20  PRINT D$;"DELETE CODE-POKES"
25  PRINT D$;"OPEN CODE-POKES"
30  PRINT D$;"WRITE CODE-POKES"
40 LINENUMBER = 7000
50  FOR PLACE = 768 TO 781
60 COUNTER = COUNTER + 1
70  IF COUNTER = 10 THEN COUNTER =
     1
80  IF COUNTER <  > 1 THEN 120
90  PRINT
100  PRINT LINENUMBER;
110 LINENUMBER = LINENUMBER + 1
120  PRINT " POKE ";PLACE;",";  PEEK
     (PLACE);" :";
130  NEXT PLACE
135  PRINT
140  PRINT D$;"CLOSE CODE-POKES"
150  END
```

When you use this program, the number in line 40 should be changed to contain the line number of your BASIC program where the POKEing program portion is to start. The FOR loop in line 50 should contain the starting and ending decimal memory locations of the machine-language routine you wish to convert.

Once you've typed in the program, RUNning it will create the text file CODE-POKES. Now use the command
EXEC CODE-POKES
to place your machine-language-POKEing program portion into any other program, beginning at the line number previously specified. The program CODE-POKES WRITER will work with either Applesoft or Integer BASIC.

MAXFILES AND INTEGER BASIC PROGRAMS

An EXEC file must be used if you want to increase MAXFILES from inside an Integer BASIC program without erasing your program. Here's how. Use the procedures described above to create an EXEC file, let's call it FILE.EX. The file FILE.EX should set HIMEM below the area that will be taken by the increased MAXFILES (595 bytes per additional file), then delete the part of the program that causes execution of the EXEC file.

FILE.EX should contain the following commands to allow for 5 files on a 48K system:

```
CLR
HIMEM: -28326
DEL 10,20
RUN
```

As shown in Table 2 of Appendix D, DOS usually sets HIMEM for a 48K system to -27136; to allow for 2 more 595 byte buffers than usual, HIMEM must be set to -27136 - (2 * 595) or -28326.

The first lines of the program would be as follows: note that what appears as CTRL-D is created by holding down the key marked CTRL while typing the letter D.

```
10 PRINT "CTRL-D EXEC FILE.EX"
20 END
30 PRINT "CTRL-D MAXFILES 5"
```

EXEC-UTIVE SESSION

The usual syntax for the EXEC command is

EXEC f

where f is the name of a sequential text file containing BASIC commands or program lines. Examples of this usage appear throughout the earlier sections of this chapter. EXEC with this syntax causes the first field of file f to be read into the Apple as if it were being typed on the keyboard. When the first field's RETURN character is "typed", the Apple attempts to execute the field's contents as a BASIC command, or enter the field's contents as a BASIC program line. The type of BASIC (Integer or Applesoft) is not changed by EXEC unless the file contains an FP or INT command. When execution has ceased on the first field, the second field of file f is read into the Apple and treated similarly. The action comes to a stop when the last field of file f has been read.

The EXEC command <u>cannot</u> be stopped by CTRL-C.

Only one EXEC file can be OPEN at any one time. If you are EXECing a file, and one of the commands thus executed is another EXEC command, the first EXEC file is immediately CLOSEd. Thereafter, it is the <u>second</u> EXEC command that is being executed.

If a file being EXECed contains a command to RUN any program, EXEC will wait patiently until the program ends. Then the next EXEC file command will be executed.

However, if a program is RUNning while an EXEC file is OPEN, any INPUT statement in the program will take the next field in the file being EXECed as its response, ignoring the keyboard. Worse yet, if that response is an immediate-execution DOS command, the command will be executed before the program continues. Results can be very confusing.

If you interrupt a RUNning Applesoft program by typing CTRL-C while an EXEC file is OPEN, the remainder of the EXEC file will usually not be executed.

If any field of an EXEC file cannot be interpreted as a valid BASIC command or program line, the message
SYNTAX ERROR
is generated, and the next field is read into the Apple. Thus, you can EXEC any text file, whether or not it contains BASIC statements (first be sure you've SAVEd any program in memory). In NON C, I, O mode, this can provide a crude but handy tool for quickly examining the contents of a text file.

The EXEC command can also be used with the Relative-field position parameter, in a way that is a bit different from POSITION's use of that parameter. The syntax for this use is
EXEC f [,Rp]
where Rp specifies that file f is to be EXEC'd starting in the p-th field of file f. Since EXEC always sets the position-in-the-file pointer to the first character of the file, so the parameter Rp always indicates the p-th field relative to the file's beginning. Thus p always corresponds to the file's actual, or absolute, field. R0 indicates that EXECing begins with the file's first field, R1 indicates EXECing begins with the second field, etc.

Note that this is different from POSITION's use of the R parameter, where R3 is a relative field only, and may indicate different actual file fields at different times in a program.

EXEC MYFILE, R1
generates an
END OF DATA
message if the R parameter specifies the second field beyond the file's end. (If the first field beyond the file's end is specified, nothing happens).

CHAPTER 8
USING RANDOM-ACCESS FILES

For a better understanding of the information presented in this chapter, it is suggested that you first read Chapter 6, on Sequential files.

RANDOM-ACCESS FILES: HOW THEY WORK

Random-access text files are like a collection of equal-sized cells in a honeycomb -- some cells may be full, others may be empty. Each "cell" is called a record. When you create a random-access file, you must specify the standard size for the records the file is to contain.

Unlike the fields in sequential files, which may be of almost any length, the records in a random-access file are of specified fixed length. The first time you WRITE to any particular record in a file, enough space is set aside on the diskette for a complete, standard-length record, whether or not the record is actually filled. So random-access files don't necessarily represent an efficient use of space. However, since these files are set up in such a regular fashion, it's fast and easy to retrieve or modify information from any part of the file -- hence the name "random-access" file.

Random-access files should be used in applications requiring fast access to various parts of the file, or where individual pieces of information in the file need to be changed fairly often. For example, a random-access file is particularly suitable for maintaining a mailing list.

Random-access files are created and retrieved in a manner very similar to that used for sequential files. The main difference is that certain commands have additional parameters: OPEN requires an L (Length of record) parameter, while READ and WRITE each use an R (Record number) parameter. Some sample programs will be presented and discussed before getting into details on creating and retrieving random-access files and how the new parameters work. More technical information about random-access text files may be found in Appendix C.

A SPECIFIC RECORD

How can you access a specific record in a random-access file? The following pair of Applesoft programs illustrates how DOS allows you to do this. The first program requests a name (N$), a telephone number (P$) and a zip code (Z$), then sends them to record 1 of a file called MAILER:

```
10  REM     MAKE MAILER
20 D$ =  CHR$ (4): REM  CTRL-D
30  INPUT "NAME:      ";N$
40  INPUT "PHONE      ";P$
50  INPUT "ZIP CODE   ";Z$
60  PRINT D$;"OPEN MAILER,L200"
70  PRINT D$;"WRITE MAILER,R1"
80  PRINT N$  PRINT P$  PRINT Z$
90  PRINT D$;"CLOSE MAILER"
```

Line 20 places a CTRL-D in the variable D$, as usual.
Lines 30 through 50 request the information to be stored.
Do not type any commas or colons in your responses.
Line 60 OPENs a file called MAILER, with 200-byte long records.
Line 70 prepares for recording information in record 1.
Line 80 sends N$, P$ and Z$ to the diskette -- since record 1 was specified in line 70, all three pieces of information are recorded in record 1, separated by RETURNs.
Line 90 CLOSEs the file.

With MON C, I, O in effect, when the program is RUN you'll see:

```
NAME:      AMY DOANS
PHONE      (425) 555-1010
ZIP CODE: 95014
OPEN MAILER,L200
WRITE MAILER,R1
AMY DOANS
(425) 555-1010
95014
CLOSE MAILER
```

Record 1 of the file MAILER can be retrieved by this program:

```
10  REM  RETRIEVE MAILER:A
20 D$ =  CHR$ (4)  REM  CTRL-D
30  PRINT D$;"OPEN MAILER,L200"
40  PRINT D$;"READ MAILER,R1"
50  INPUT N1$,P1$,Z1$
70  PRINT D$;"CLOSE MAILER"
```

When RUN with MON C, I, O, you'll see the following. As usual, the pair of question marks indicates an INPUT with more than one response.

```
OPEN MAILER,L200
READ MAILER,R1
?AMY DOANS
??(425) 555-1010
??95014
CLOSE MAILER
```

And here is a slightly different program to retrieve record 1 of MAILER.

```
10  REM   RETRIEVE MAILER.B
20 D$ = "". REM  QUOTES CONTAIN
     CTRL-D
30  PRINT D$;"OPEN MAILER,L200"
40  PRINT D$;"READ MAILER,R1"
50  INPUT N1$
```

```
60  INPUT P1$
70  INPUT Z1$
80  PRINT D$;"CLOSE MAILER"
90  END
```

MULTIPLE RECORDS

The program that created the random-access file MAILER wrote to a single record in the file, saving three different pieces of information separated by RETURNs. The next program demonstrates writing to several records: in particular, record numbers 12 through 15 of a random-access file called RA-FILE.

```
5  REM  MAKE RA-FILE
10 D$ =  CHR$ (4): REM  CTRL-D
20  PRINT D$;"OPEN RA-FILE"
30  PRINT D$;"DELETE RA-FILE"
40  PRINT D$;"OPEN RA-FILE, L30"
50  FOR I = 12 TO 15
60  PRINT D$;"WRITE RA-FILE, R";I
70  PRINT "NAME ADDRESS ";I
80  NEXT I
90  PRINT D$;"WRITE RA-FILE, R13"
100  PRINT "DOS VERSION 3.2"
110  PRINT D$;"CLOSE RA-FILE"
```

Line 10 sets D$ to CTRL-D.
Lines 20 and 30 make sure RA-FILE is a new file
Line 40 OPENs the file RA-FILE, whose records will each be 30 bytes in length.
Lines 50 through 80 create a loop that WRITEs the information
NAME ADDRESS
followed by the record number, for records 12 through 15
Note that you must specify each record in a new WRITE command, before having PRINT send characters to that record.
Lines 90 and 100 change the information in record 13 to the text given in line 100's PRINT command.
Line 110 CLOSEs the random-access file RA-FILE.

If MON C,I,O is in effect when the program is RUN, you'll see the following:

```
OPEN RA-FILE
DELETE RA-FILE
OPEN RA-FILE, L30
WRITE RA-FILE, R12
NAME ADDRESS 12
WRITE RA-FILE, R13
NAME ADDRESS 13
WRITE RA-FILE, R14
NAME ADDRESS 14
```

```
WRITE RA-FILE, R15
NAME ADDRESS 15
WRITE RA-FILE, R13
DOS VERSION 3.2
CLOSE RA-FILE
```

In a similar fashion, you can READ information from a selected record or records of a text file. The next program retrieves records 12 through 15 of the file called RA-FILE, trying, on line 60, to find which record(s) contains the letters "DOS" as the first three characters.

```
5  REM  RETRIEVE RA-FILE
10 D$ = CHR$ (4)  REM  CTRL-D
20  PRINT D$;"OPEN RA-FILE, L30"
30  FOR J = 12 TO 15
40  PRINT D$;"READ RA-FILE, R";J
50  INPUT A$
60  IF  LEFT$ (A$,3) = "DOS" THEN
      PRINT "RECORD "; J;" WAS CHA
    NGED."
70  NEXT J
80  PRINT D$;"CLOSE RA-FILE"
```

Line 10 sets up CTRL-D in D$.
Line 20 OPENs the text file RA-FILE, whose records are 30-bytes long (that's what we specified when the file was created in an earlier program, remember?).
Lines 30 through 70 READ records 12 through 15 of RA-FILE. Note that you must specify each record in a new READ command, before a subsequent INPUT will read characters from that record. In line 50, each record comes in from the disk as an ASCII string terminated by a RETURN. Line 60 checks the 3 leftmost characters of the INPUT string A$ from record r, to see if the word "DOS" is there. If it is, the message "RECORD r WAS CHANGED." is printed and the search continues.
Line 80 closes the file.

Here's what you'll see when you RUN the program, if MOW C, I, O is in effect:

```
OPEN RA-FILE, L30
READ RA-FILE, R12
?NAME ADDRESS 12
READ RA-FILE, R13
?DOS VERSION 3.2
RECORD 13 WAS CHANGED.
READ RA-FILE, R14
?NAME ADDRESS 14
READ RA-FILE, R15
?NAME ADDRESS 15
CLOSE RA-FILE
```

Notice that when the file was retrieved only records that had been written to were examined. If you had asked for record 8 in RA-FILE, you would have received the
END OF DATA
message, since no information had been written to that record of the file. Similarly, had you tried to INPUT more than one field from any of the existing records you would have been given the same message: each of records 12 through 15 contains only one field.

A DEMONSTRATION: THE RANDOM PROGRAM

Last but by no means least, the System Master diskette contains a program called RANDOM that uses a random-access text file to demonstrate a small inventory control scheme. And by small we mean small: the program can handle at most 9 parts. This keeps the program simple. The Apple, of course, is capable of handling thousands of parts in an inventory.

First the program copies itself and the random-access text file APPLE PROMS used to keep track of the inventory, then it automatically RUNs the program for you. You can list one or all items in the inventory. You can also change items, either one at a time or all at once. Here's how it works. Remember to press the RETURN key each time you complete a response.

1) From the System Master,
   RUN RANDOM
   and you should see the message

```
THIS DEMONSTRATION WILL NOT EXECUTE ON
A WRITE-PROTECTED DISKETTE SUCH AS
YOUR DOS SYSTEM MASTER (VERSION 3.2)
FOR YOUR CONVENIENCE, PROVISIONS HAVE
BEEN MADE TO COPY THIS PROGRAM AND IT'S
DATA TO ANOTHER DISKETTE

DO YOU WISH TO DO THIS NOW? (Y OR N) Y
```

   If you type N for "no" in response to the above message, you'll find yourself back in Applesoft.

2) Press Y FOR "yes". You'll see the message

```
NOW READING DATA
```

   Followed by the message

```
INSERT AN INITIALIZED DISKETTE, THEN
PRESS THE RETURN KEY TO BEGIN TRANSFER
```

3) Remove the System Master diskette, and place a non-write-protected diskette in the drive, then press the RETURN key. You'll perhaps catch a glimpse of the message

and then the program will begin execution.

4) Now you should see this:

```
                APPLE PROMs
COMMAND          NUMBER
-------          ------
LIST               1
CHANGE             2
EXIT               3
CHOOSE NUMBER (1 - 3) 1
```

Press 1 and you should see this message:

```
PART NUMBER 1-9 (0=ALL) 0
```

5) Press 0, to get a list of all "parts" in this "inventory system" and you'll see

```
PART#  NAME             SIZE    IN STOCK
-----  ----             ----    --------
  1    PARALLEL PRINT   256      500
  2    COMMUNICATIONS   256      1250
  3    (NOT AVAILABLE)  256      0
  4    (NOT AVAILABLE)  256      0
  5    DISK BOOT        256      432
  6    STATE MACHINE    256      460
  7    SERIAL PRINTER1  256      878
  8    SERIAL PRINTER2  512      741
  9    CENTRONICS       256      1290

PRESS THE RETURN KEY TO CONTINUE.
```

When you're ready to return to the list of options, press the RETURN key.

6) Try out the various program options. Choice 1 allows you to list parts by part number, one at a time, as well as all at once.

Choice 2 allows you to change any or all part names and descriptions. For example, suppose part 3 should be named COSMIC GLUE, size 56, with 1234 in stock. Here's how to revise the entry for part 3:

select option 2, CHANGE
select part number 3
the old part name is displayed, with the cursor at its start, to allow you to enter the new name; when you

press the RETURN key the cursor will move to the right and perform similarly for part size and quantity
to use the currently existing name or size or quantity, just press the RETURN key by itself.

Choice 3 will stop the program.

WRITE-ING AND READ-ING RANDOM-ACCESS TEXT FILES

When used with random-access files, the CLOSE command works exactly as it does with sequential files (see "OPENing and CLOSEing Sequential Files" in Chapter 6). However, the syntax for OPEN has an additional parameter, the L parameter, which is required.

```
OPEN f [,Lj [,Bs] [,Dd] [,Vv]
```

The "L" stands for "Length-of-record"; the number j indicates how many bytes (characters and digits) are to be allotted to each record in the random-access file you're creating (or, if you're retrieving a file, the number that were allotted when the file was created). If the L option is omitted, j is assigned the default value of 1. The number j must be in the range 1 through 32767.

When you OPEN a file prior to READing, if you specify a different Length parameter, than you specified when you OPENed prior to WRITEing the file, DOS will blindly use the new Length parameter to calculate record positions within the file. You will have to keep detailed written documentation on the structure and contents of your files (some programmers keep such information in record #0 of the file). It's helpful to always include the Length parameter in each file's name, with such names as

```
RANDFILE.L128
STOCKLIST.L100
DIRECTORIES(L50)
```

There is no way to find the length of a record in a random-access file: you must make this information part of your documentation.

Records should never be longer than the number of bytes specified by the L parameter: records may be partially over-written or combined with confusing results.

WRITE and READ each have an R parameter, to be used when creating or retrieving particular records in random-access files.

```
WRITE f [,Rr]
 READ f [,Rr]
```

```
Examples: WRITE LEGIBLY, R3,
          READ FAST, R13
```

The Rr (Record) parameter is used to create (with WRITE) or retrieve (with READ) the rth record of the file. The default value of r is Ø, specifying the first record of a file.

Using CTRL-C to stop a READ in Applesoft causes a string of REENTERs to be generated: press the RESET key instead.

In some respects, each separate record in a random-access text file may be treated as a short sequential file. WRITE and READ can be used with a Byte parameter in addition to their R parameter. The Byte parameter specifies the beginning byte of the specified record, for the next PRINT (after WRITE) or INPUT or GET (after READ).

```
WRITE f [,Rr] [,Bb]
 READ f [,Rr] [,Bb]
```

If specified, the B (Byte) parameter causes WRITEing (or READing) to begin at the b-th byte of the specified record. The default value of b is Ø, the first byte of a record. The R parameter may specify a position in the record either before or after the current position-in-the-file pointer. Using the B parameter necessitates a thorough, detailed, byte-by-byte knowledge of the contents of each record in the file.

Once READ or WRITE has moved the position-in-the-file pointer to a particular record, POSITION can also be used to move the pointer ahead (only) to further relative-field positions within the record. However, POSITION cancels either WRITE or READ mode (without changing the position-in-the-file pointer), so another WRITE or READ command (this time with no parameter) is necessary to re-instate that mode.

Details on how information is stored on the diskette in general, and in random-access files in particular, may be found in Appendix C.

CHAPTER 9
USING MACHINE LANGUAGE FILES

MACHINE LANGUAGE FILES

DOS allows you to store on diskette, and retrieve from diskette, the information in your Apple II's memory. You have already seen the DOS commands SAVE, LOAD and RUN: these commands deal with the contents of Apple's program memory, interpreted as commands in BASIC programs. The DOS commands discussed in this chapter -- BSAVE, BLOAD and BRUN -- perform similar functions, but they deal with the contents of any portion of Apple's memory, in its uninterpreted, raw binary-and-hexadecimal form.

The B before each of the following commands stands for a Binary file; a B also precedes the name of binary files in the CATALOG. A binary file is just an exact, bit-for-bit copy of the information that was stored in a specified range of Apple memory locations. Those locations may have contained a machine-language program, binary data, or a bit-mapped "picture" from Apple's high-resolution graphics screen.

BSAVE

The BSAVE command creates a file named f and stores all the contents of a segment of memory. The syntax is

BSAVE f ,Aa, Ll [,Ss] [,Dd] [,Vv]

where as usual the S, D, and V parameters stand for slot number, drive number, and volume number. Note that the A and L parameters are not optional.

The A parameter specifies the starting Address (in either decimal or hexadecimal code) of the memory portion to be stored on diskette. A dollar sign ($) must precede an address expressed in hexadecimal. If the A parameter is less than 0 or greater than 65535, a

SYNTAX ERROR

message is displayed. Therefore, equivalent negative addresses may not be used with this command. Within the range 0 through 65535, no error message is generated if the A parameter specifies a starting memory address that does not correspond to actual, installed memory chips. In practice, it is not useful to specify an A parameter greater than the maximum memory address in your Apple (49151 or $BFFF on a 48K system).

The L parameter specifies the Length, in bytes, of the memory portion to be stored. If the L parameter is less than 0 or greater than 65535, a

SYNTAX ERROR

message is generated. If the L parameter is 0 or in the range 32768 through 65535, a

RANGE ERROR

message is generated. 32767 is the greatest number of bytes that can be stored in a single field on the diskette. If you wish to store more than 32767 memory locations, use two BSAVEs. Within the range 1 through 32767, no error message is generated if the L parameter specifies a range of memory addresses, not all of which correspond to actual, installed memory chip. In practice, it is not useful to specify a range of memory addresses extending beyond the maximum memory address in your Apple (49151 or $BFFF on a 48K system).

These examples each create a file named PICTURE containing an image of the second high-resolution graphics area of the Apple's memory. They are operationally identical, but their starting address and length parameters are given in different forms.

```
BSAVE PICTURE, A$4000, L$2000
BSAVE PICTURE, A16384, L8192
BSAVE PICTURE, A16384, L$2000
BSAVE PICTURE, A$4000, L8192
```

BLOAD

The BLOAD command returns the contents of a Binary file to your Apple II's memory. BLOAD does not erase a BASIC program in memory, unless the data is BLOADed into the particular portion of memory containing your program.

The syntax is
BLOAD f [,As] [,Ss] [,Dd] [,Vv]
where the S, D, and V parameters are as usual. If the A parameter is used, then the Binary file's contents replace a portion of the existing contents of Apple's memory, beginning at address a. If the A parameter is not used, the file's contents are returned to the same Apple memory locations whose contents were originally BSAVEd. See BSAVE for a complete discussion of the A parameter.

Assume the binary file PICTURE contains a high-resolution picture. Either of these examples places the picture into the first high-resolution graphics area of the Apple's memory:
BLOAD PICTURE, A8192
BLOAD PICTURE, A$2000
Either example also clobbers the RAM version of Applesoft.

Note: a machine-language program may no longer be executable if it is moved to a memory location different than the one from which it was saved.

BRUN

The syntax of the BRUN command is the same as for BLOAD:
BRUN f [,As] [,Ss] [,Dd] [,Vv]
The Binary file f should be a machine-language program.

First BRUN does a BLOAD. If the A parameter is given, the file's contents are placed into Apple's memory beginning at location a. If the A parameter is not used, the file's contents are returned to the same Apple memory locations whose contents were originally BSAVEd. See BSAVE for a complete discussion of the A parameter.

After BLOADing the file, BRUN does a machine language jump (JMP) to location a. If the file was a machine-language program, this begins execution of that program.

THE RWTS SUBROUTINE

Normally, user access to and from the DISK II is restricted to the use of DOS. However, another method of accessing the DISK II is available to machine language programmers. You may skip this section if you're not familiar with machine language.

The DISK II can be accessed directly from machine language through the use of the RWTS subroutine, which is part of the DOS. The "RWTS" stands for "Read or Write a Track and Sector". In the following explanation, any numbers preceded by $ are hexadecimal numbers.

Every diskette initialized by the DISK II drive is separated into 35 tracks, numbered 0 to 34. These tracks may be thought of as grooves on a phonograph record, except that they are not connected with each other. Basically, the tracks are arranged in separate concentric circles, with the large hole in the center of the diskette forming the common center of the circles. Track 0 is on the outer edge of the diskette, while track 34 is nearest the center. The disk drive has a "head" that acts very much like the needle on a record player, except that the head on the disk drive is magnetic. This head moves to different tracks on the diskette, where it either reads information off of the diskette, or writes information onto the diskette.

Each track on the diskette consists of 13 sectors. Sectors are pre-defined groupings on each track, that allow the user to work with single blocks of 256 bytes, rather than with the entire 3328 bytes that fit on one track. The sectors within a track are individually numbered, consecutively, 0 to 12 around the diskette. As the diskette spins, each sector will pass underneath the head, at which time the head may write to or read from that sector. Each sector consists of two portions: the address field and the data field. The address field contains information concerning which track the head is on, which sector is about to spin past the head, and the volume number of the diskette. The data field contains an encrypted form of the actual 256 bytes of data which were stored on that sector.

The "Read or Write a Track and Sector" subroutine (referred to as the RWTS subroutine), allows the user to write information to, or read information from, any track and sector on the diskette, via machine language. In order to use the RWTS subroutine, the user must first create an IOB (Input/Output control Block) table, and an accompanying "Device Characteristics Table". The IOB tells the RWTS subroutine which slot and drive number the disk drive will be in, which volume number to expect on the diskette, which track and sector to access, and whether to read from or write to the diskette. The Device Characteristics Table provides some information to the RWTS subroutine that is necessary to operate the Apple DISK II.

To use the RWTS subroutine, the user must set up the IOB and the Device Characteristics Table somewhere in memory. A "controlling subroutine" must be written and stored in memory. The subroutine must JSR to the starting address of the RWTS subroutine (at location $3D9). When the RWTS subroutine is jumped to, the A and Y registers must contain the address of the starting location of the IOB. The A register must contain the high

address byte, and the Y register the low address byte. The format of the IOB is given in Table 3, at the end of this section. Table 4 gives the format of the Device Characteristics Table.

Here is an example of how to use the RWTS subroutine. The example IOB, Device Characteristics Tables, and a controlling subroutine will all be loaded into memory just after location $C00.

The following controlling subroutine will load the A and Y registers with the address of the starting location of the IOB, and then jump to the RWTS subroutine.

```
$C00-   A9 0C       LDA   #$0C   Load A register with $0C
$C02-   A0 0A       LDY   #$0A   Load Y register with $0A
$C04-   20 D9 03    JSR   $03D9  Jump to the RWTS subroutine
$C07-   60          RTS
$C08-   00          BRK
```

The following IOB is one that you would use to access slot 6, drive 1, to write 256 bytes of memory starting at location $2000, onto track 18, sector 6 of the diskette:

Location	Code	Purpose
$C0A	01	IOB type indicator, must be $01
$C0B	60	Slot number times 16
$C0C	01	Disk drive number
$C0D	00	Expected volume number
$C0E	12	Track number
$C0F	06	Sector number
$C10	20	Low-order byte of Device Characteristics Table
$C11	0C	High-order byte of Device Characteristics Table
$C12	00	Low-order byte of data buffer starting address
$C13	20	High-order byte of data buffer starting address
$C14	00	Unused
$C15	00	Unused
$C16	02	Command code, $02 = write
$C17	00	Error Code
$C18	00	Actual volume number
$C19	60	Previous slot number accessed
$C1A	01	Previous drive number accessed

The following Device Characteristics Table must be included, we'll place it at location $C20, just after the IOB. Locations $C10 and $C11 in the IOB above point to the address of the Device Characteristics Table's starting location.

Location	Code	Purpose
$C20	00	Device type code (put a $00 here)
$C21	01	Number of phases per track (put a $01 here)
$C22	EF	Time count (put a $EF here)
$C23	D8	Time count (put a $D8 here)

When you have loaded the IOB at $C0A, the Device Characteristics Table at $C20, and the controlling subroutine to load the A and Y registers at $C0D, then
C0DC
or
CALL 3072
will cause the entire routine to execute.

TABLE 3: FORMAT OF IOB

Byte#	Name	Purpose
1	IBTYPE	Tells the RWTS subroutine what type of IOB this is. Should be a $01. No other type codes are currently defined.
2	IBSLOT	Must contain the number of the slot times 16, in which the disk drive's controller card is located. For example, if you want to access slot #6, the value $60 must be stored in this location.
3	IBDRVN	Must contain the number of the disk drive to be accessed -- either $01 or $02.
4	IBVOL	The volume number of the diskette to be accessed must be stored here. Volume $00 will match the volume number assigned to any diskette.
5	IBTRK	The number of the track (0 to 34) to be accessed is stored here. Must be within the range $00 to $22.
6	IBSECT	The number of the sector (0 to 12) to be accessed is stored here. Must be within the range $00 to $0C.
7&8	IBDCTP	These two bytes must contain the address of the starting location of the Device Characteristics Table (see below). Byte 7 must contain the low-order byte of the address, and byte 8 must contain the high-order byte.
9&10	IBBUFP	Bytes 9 and 10 must contain the address of the starting location of the "data buffer". The data buffer is a 256-byte long section of memory upon which the RWTS subroutine will operate. If you are writing onto the diskette, the information in the data buffer will be

written onto the diskette. If you are reading from the diskette, the information that comes off of the disk will be stored in memory at the location of the data buffer. 256 bytes is both the minimum and the maximum amount of information that can be read or written by the RWTS subroutine.

11&12	-----	Unused
13	IBCMD	In this byte is stored the command code that tells the RWTS subroutine exactly what to do. The values that can be stored in byte 13 are: $ØØ -- Null command. Does nothing but start the disk drive and position the head. $Ø1 -- Read the entire 256 bytes stored on the diskette at the specified track and sector, and store them in memory at the location of the data buffer. $Ø2 -- Write the next 256 bytes stored in memory at the location of the data buffer on to the diskette at the specified track and sector. $Ø4 -- Format the diskette. When a diskette is formatted, self-synchronizing nibbles are written on every track and sector on the diskette. Because all of the diskette is formatted, the values in bytes 5 and 6 are ignored. All of a formatted diskette is available for use; there is no DOS, or anything stored on the diskette until the user puts something there.
14	IBSTAT	This location will contain the code number for any error that may be encountered during execution. If the RWTS subroutine returns with the carry flag clear, no error has occurred. If it returns with the carry flag set, this byte indicates what type of error has occurred. $1Ø -- Diskette is write-protected, and cannot be written to. $2Ø -- Volume mismatch error. The volume number of the diskette found was different than the volume specified in byte 4. $4Ø -- Drive error. Something unusual is happening. $8Ø -- Read error. The RWTS routine was, after 48 repeated attempts, unable to read either the address field or the data field. If the data field for the specified sector has never had anything written on it, then a read error will result, because there is nothing to read.
15	IBSVOL	The volume number of the diskette that is actually found will be stored in this location.

TABLE 3: FORMAT OF IOB [continued]

Byte#	Name	Purpose
16	IOBPSN	This byte must contain the slot number times 16 of the slot that was accessed most recently. For example, if you previously accessed a disk drive in slot 5, store the value $50 here. If there is no controller in the specified slot, the disk will hang.
17	IOBFDN	This byte must contain the number of the disk drive that was accessed most recently -- a $01 or $02.

Table 4: FORMAT OF DEVICE CHARACTERISTICS TABLE

Byte#	Name	Purpose
1	DEVTPC	Device type code, telling what type of device this is. A $00 should be stored in this byte for use with a DISK II.
2	PPTC	Number of phases per track. A $01 should be stored here.
3&4	MONTC	Motor on time count complemented, in 100 micro-second intervals. A $EF should be in byte 3, and a $D8 in byte 4, for use with a DISK II.

CHAPTER 10
INPUT, OUTPUT AND CHAINING

SELECTING I/O DEVICES: IN#, PR# AND OTHER STUFF

There are various ways in which information can be used as input or output for your Apple computer. Very often the keyboard serves as a source of input. Usually the Apple uses a TV screen for output, but any accessory or peripheral connected to a controller in one of the seven Apple accessory slots can be used for input or output using the IN# and PR# commands.

Examples:

IN# 6 obtains subsequent input from the device controlled from slot #6. Note: if slot #6 contains a disk controller card, this command will cause DOS to be booted. If no device is in slot #6, the system may "hang". Press the RESET key to recover.

IN# Ø obtains subsequent input from the keyboard (not slot #Ø), instead of a peripheral device.

PR# 1 transfers output to the device controlled from slot #1, usually the printer. Note: if no device controller card is installed in slot #1, the system may "hang" and you'll have to press the RESET key to recover.

PR# Ø returns output to the TV screen (not to slot #Ø).

The syntax for the commands is

IN# e

or

PR# e

where e specifies the slot to use. What happens depends on e:

value of e	result
less than Ø	SYNTAX ERROR
Ø	re-establishes usual device (for IN#, input from the keyboard, for PR#, output to the TV screen)
1 through 7	transfers to device controlled from the specified slot (boots DOS if a disk controller card is in that slot)
8 through 16	SYNTAX ERROR in deferred-execution mode; the system hangs in immediate-execution mode
17 through 65535	RANGE ERROR
greater than 65535	SYNTAX ERROR

The command IN# Ø re-establishes input from the keyboard; PR# Ø re-establishes output to the TV screen.

With DOS in effect, the IN# and PR# commands may be used in immediate execution mode in the usual way (see your BASIC manuals). But when they

are issued by lines in a program, IN# and PR# must take the form of DOS commands such as

```
10  D$ = "": REM  CTRL-D
20  PRINT D$; "PR# 1"
30  PRINT D$; "IN# 2"
```

When DOS is not in effect, the IN# and PR# commands set the contents of the Apple Monitor Input and Output registers to select the desired input and output devices.

When DOS is in effect, the contents of the Apple Monitor Input and Output registers are set to select DOS, while the contents of the DOS Input and Output registers are set to select the desired input and output devices. The following paragraphs describe what happens each time a character leaves or enters the Apple.

When the Apple sends an output character, the Apple Monitor Output register directs that character to DOS. If the character is to be sent on (because it is not part of a DOS command), DOS does a fast two-stage switch:

1. First, DOS replaces the contents of the Apple Monitor Input and Output registers with the contents of the DOS Input and Output registers. Then it sends the character out to the device now selected by the contents of the Apple Monitor Input and Output registers.

2. Next, DOS re-connects itself by again placing the pointers to DOS in the Apple Input and Output registers.

Similarly, each time the Apple asks for an input character, the Apple Monitor Input register directs that request to the DOS. Once again, DOS does its fast two-stage switch:

1. First, DOS replaces the contents of the Apple Monitor Input and Output registers with the contents of the DOS Input and Output registers. Then it obtains an input character from the device now selected by the Apple Monitor Input and Output registers.

2. Next, DOS re-connects itself by again placing the pointers to DOS in the Apple Input and Output registers.

When DOS is in effect, DOS intercepts all input characters from the input device before they reach Applesoft or Integer BASIC or the Monitor. That is why IN# and PR#, when typed on the keyboard as immediate-execution commands, can be trapped and used by DOS to change the DOS Input and Output registers.

Similarly, DOS intercepts all output characters from the Apple before they reach the output device (but after they have affected the Apple Monitor Input and Output registers). That is why IN# and PR#, if issued from within a program but not in PRINTed DOS commands, can disconnect DOS by changing the Apple Monitor Input and Output registers before the commands ever get to DOS. Because the entire contents of the Apple Monitor Input

and Output registers are replaced each time DOS attempts to send or receive a character, DOS will usually re-connect itself if it was not disconnected at both Input and Output registers simultaneously.

If you execute a PR# command from within a program, with a program line such as
50 PR# 1
then DOS will be partially disconnected and unable to interrept subsequent output. DOS is still connected for input, and the next attempt to obtain any input character will cause DOS to re-connect itself for both input and output.

The same situation occurs with the use of IN# inside programs when DOS is in effect. A program line such as
60 IN# 1
will disconnect DOS for subsequent input. DOS is still connected for output, and the next attempt to send out a character (even a return or a prompt character) will cause DOS to re-connect itself for both input and output. To avoid such conflicts and allow DOS to manage the Input and Output registers, issue PR# and IN# commands in immediate-execution mode, or as DOS commands in program lines such as those mentioned earlier:

```
10  D$ = "": REM  CTRL-D
20  PRINT D$; "PR# 1"
30  PRINT D$; "IN# 2"
```

The CTRL-D character tells DOS that the following output characters are a DOS command.

TABLE 1: APPLE MONITOR INPUT AND OUTPUT REGISTERS

Monitor Input Register: Locations 56-57 ($38-$39)

When Register contents are set by	To the value	Then subsequent input comes from
RESET	-761	Monitor Input Routine
Ø CTRL-K [Note 1]	($FD1B)	from Apple keyboard
IN#Ø [Note 2]		
s CTRL-K [Note 1]	49152 + s*256	Slot #s
IN#s [Note 2]	($Cs00)	If slot #s contains disk
[where s>Ø]		controller, then boot DOS
DOS boot	-8626 + Top of mem.	DOS
	(-$21B2 + $Top of mem.)	

Monitor Output Register: Locations 54-55 ($36-$37)

When Register contents are set by	To the value	Then subsequent output goes to
RESET	-328	Monitor Output Routine
Ø CTRL-P [Note 1]	($FDF0)	to TV screen
PR#Ø [Note 2]		
s CTRL-P [Note 1]	49152 + s*256	Slot #s
PR#s [Note 2]	($Cs00)	If slot #s contains disk
[where s>Ø]		controller, then boot DOS
DOS boot	-8577 + Top of mem.	DOS
	(-$2181 + $Top of mem.)	

Note 1. The commands s CTRL-K and s CTRL-P are Monitor commands. To type CTRL-K (which does not appear on the TV screen), type K while holding down the CTRL key.

Note 2. When DOS is in effect, this command will effect the contents of the Apple Monitor register only if the command is issued as an instruction in a stored program and not in a PRINT CTRL-D instruction.

Note 3. In addition to the commands mentioned in Table I, directly POKEing appropriate values into the Apple Monitor register locations can also be used to select input and output devices, or to re-connect a disconnected DOS.

TABLE 2: DOS INPUT AND OUTPUT REGISTERS

DOS Input Register

When Register contents are set by	To the value	Then subsequent input comes from
DOS boot RESET 3D0G IN#0 [Note 4] PRINT D$;"IN#0" [Note 5]	-741 ($FD1B)	Monitor Input Routine from Apple keyboard
IN#s [Note 4] PRINT D$;"IN#s" [Note 5] [where s>0]	49152 + s*256 ($Cs00)	Slot #s If slot #s contains a disk controller, then reboot DOS

DOS Output Register

When Register contents are set by	To the value	Then subsequent output goes to
DOS boot RESET 3D0G PR#0 [Note 4] PRINT D$;"PR#0" [Note 5]	-528 ($FDF0)	Monitor Output Routine to the TV screen
PR#s [Note 4] PRINT D$;"PR#s" [Note 5] [where s>0]	49152 + s*256 ($Cs00)	Slot #s If slot #s contains a disk controller, then reboot DOS

Note 4. When DOS is in effect, this command will not affect the contents of the DOS Input and Output registers if the command is issued as an instruction in a stored program and not in a PRINT CTRL-D instruction. If a program line such as
120 PR#3
is executed, the contents of the Apple Monitor Output register will be changed, leaving DOS partially disconnected until the next input.

Note 5. In this command, it is assumed that the string-variable named D$ has been assigned the character control-D, or CTRL-D. This character, which does not appear on the screen, is produced by typing D while holding down the CTRL key.

Note 6. No matter what input or output device is selected by the DOS Input and Output registers, input can also be received from the disk and output can be sent to the disk.

Note 7. In addition to the commands in Table II, directly POKEing the appropriate values into the DOS Input and Output register locations can also be used to select input and output devices. However, the specific memory locations of the DOS Input and Output registers change with different system memory sizes and with different versions of DOS. For this reason, a special procedure exists for changing the contents of the DOS Input and Output register locations. It is a two step procedure:

a) Change the Apple Monitor Input and Output register locations to the values you wish the DOS Input and Output registers to contain. (This may be done by directly POKEing the Apple Monitor register locations or by executing IN# and PR# non-DOS instructions in a stored program.)

b) CALL 1002 (from the Monitor, you would type $3EA6).

After this CALL, DOS will be re-connected via the Apple Monitor registers, and the previous contents of the Apple Monitor Input and Output registers will appear in the DOS Input and Output register locations. This CALL can also be used to re-connect DOS any time your program needs to disconnect DOS for awhile. See the program on page 151 for an example using this technique.

Note 8. The Monitor commands a CTRL-K or a CTRL-P , when typed on the keyboard, are not recognized by DOS: they affect the Apple Monitor Input or Output registers directly.

INTEGER BASIC CHAIN

Certain applications are most easily implemented by using a series of two or more programs which are LOADed and RUN sequentially. In such circumstances, the second program often needs to use the values of variables and arrays developed by the first program. The usual RUN command erases the first program's variables and arrays when it loads the second program. In Integer BASIC (but not Applesoft) the DOS command CHAIN allows you to load and run a second program without erasing the first program's variables and arrays.

Suppose you've been using an Integer BASIC program called PART ONE. The command
CHAIN PART TWO
will load and run the Integer BASIC program called PART TWO without clearing the values of any variables used in the program PART ONE. The CHAIN command may be issued in immediate-execution mode as shown above, or from within the last lines of the PART ONE program as a DOS command:

```
20010  D$="": REM  CTRL-D
20020  PRINT D$; "CHAIN PART TWO"
```

The syntax for the command is familiar:
CHAIN I [,Ss] [,Dd] [,Vv]

APPLESOFT CHAIN

The CHAIN command works only with Integer BASIC, but if you do not need to pass variables, it is easy to link Applesoft programs to load and run in sequence. In the first program, just include a last line such as
```
20000  PRINT CHR$(4); "RUN PART TWO"
```
When this line is executed, it will start up the second program (where the second program is named PART TWO). In the process, the first program and all its variables are erased.

A different procedure must be used in order to load and run a series of Applesoft programs without erasing earlier values of variables and arrays. To chain in Applesoft, you will need to use the machine-language program called CHAIN that is on the DOS version 3.2 System Master diskette.

To chain from a program called PART ONE to a program called PART TWO, you must have the CHAIN program on the same diskette with the program PART TWO (see next page for instructions). Then, simply insert these two lines as the last two lines to be executed in the PART ONE program:
```
20000  PRINT CHR$(4); "BLOAD CHAIN, A$20"
20010  CALL 520"PART TWO"
```
The two lines may use any line numbers, but they should come one after the other in the program, as indicated. The first line loads the Applesoft chaining ability into the computer. The second line actually does the chaining (but see next page, for warning).

There must be no space in the third line between the CALL address 520 and the following quotation mark. The CALL address must not be given in hexadecimal.

If you have Applesoft on the firmware ROM card, you can copy the CHAIN program onto another diskette as follows. First place the CHAIN program into Apple's memory, with the command
BLOAD CHAIN, A2056
Then save it on the desired diskette, with the command
BSAVE CHAIN, A2056, L456

If you are using RAM Applesoft (on diskette), you can copy the CHAIN program onto another diskette as follows. First place the CHAIN program into Apple's memory, with the command
BLOAD CHAIN, A12296
Then save it on the desired diskette, with the command
BSAVE CHAIN, A12296, L456

Note that neither Address parameter for copying CHAIN is the same as the Address parameter for actually using CHAIN.

APPENDIX A
FILE TYPES USED WITH DOS COMMANDS

Unless otherwise indicated, DOS commands may be used either in immediate-execution mode or in deferred-execution mode (within a program). However, some text file commands (e.g. READ and WRITE) must be used in deferred-execution mode.

Most DOS commands refer to a named file. A file may be a text (data) file, or a program in Integer BASIC, APPLESOFT or Machine Language. The tables below indicate which file types may be used by each command. The first table lists the commands alphabetically; the second table groups them by associated file type. The commands CATALOG, FP, INT, MAXFILES, MON, NOMON, PR# and IN# are not included because they do not explicitly refer to named files.

FILE TYPE USE, LISTED BY DOS COMMAND

DOS Command Used Files:	Integer BASIC Program File	Applesoft BASIC Program File	Sequential Access Text File	Random Access Text File	Machine Language Binary File
APPEND			x		
BLOAD					x
BRUN					x
BSAVE					x
CHAIN	x				
CLOSE			x	x	
DELETE	x	x	x	x	x
EXEC			x		
INIT	x	x			
LOAD	x	x			
LOCK	x	x	x	x	x
OPEN			x	x	
POSITION			x		
READ			x	x	
RENAME	x	x	x	x	x
RUN	x	x			
SAVE	x	x			
UNLOCK	x	x	x	x	x
VERIFY	x	x	x	x	x
WRITE			x	x	

Note: use these commands only in deferred execution mode:
APPEND, OPEN, POSITION, READ, WRITE

FILE TYPE USE, LISTED BY FILE TYPE

Integer BASIC files only
CHAIN

Integer BASIC or APPLESOFT files
INIT
LOAD
SAVE
RUN

Sequential Text files only
APPEND
EXEC
POSITION

Either Sequential Text Files or Random-Access Text Files
OPEN
CLOSE
READ
WRITE

Machine Language files only
BLOAD
BRUN
BSAVE

All Types of Files
DELETE
LOCK
UNLOCK
RENAME
VERIFY

Note: these commands must be used in deferred-execution mode:
APPEND, OPEN, POSITION, READ, WRITE

APPENDIX B
DOS MESSAGES

When DOS detects an error connected with disk usage, it normally displays a message describing the error and stops any program that is running. These messages are in addition to the usual messages generated by Applesoft or Integer BASIC. DOS messages can be distinguished from those of Applesoft or Integer BASIC as follows:

An Applesoft message, such as
?SYNTAX ERROR
is preceded by a question mark.

An Integer BASIC message, such as
*** SYNTAX ERR
is preceded by three asterisks.

A DOS message, such as
SYNTAX ERROR
is preceded by no character at all.

A DOS message appears exactly the same, whether you are in Applesoft, Integer BASIC or the Monitor at the time the message is generated.

If a DOS message occurs when you are using the Monitor, the system is reset to the type of BASIC from which you entered the Monitor.

By using Applesoft's ONERR GOTO command (see the Applesoft manual), you can create Applesoft error-handling routines that deal with DOS messages which would normally interrupt your program. When a DOS error occurs following an ONERR GOTO command in an Applesoft program, a code number for the type of error is stored in decimal memory location 222. This is the same memory location in which Applesoft stores the code for an Applesoft error message. The command
Y = PEEK(222)
sets the value of Y to the Applesoft ONERR GOTO code corresponding to the error that caused an Applesoft ONERR GOTO jump to occur.

DOS messages and their corresponding Applesoft ONERR GOTO codes are shown below, with the most common cause of each message. Each of the messages is then discussed in greater detail, with a more comprehensive list of causes and cures.

ONERR GOTO CODES

code	DOS message	Most common cause
1	LANGUAGE NOT AVAILABLE	Applesoft not on diskette
2,3	RANGE ERROR	Command parameter too large
4	WRITE PROTECTED	Write-protect tab on diskette
5	END OF DATA	READing beyond end of text file
6	FILE NOT FOUND	File misspelled, or not on diskette
7	VOLUME MISMATCH	Wrong Volume parameter

ONERR GOTO

code	DOS message	Most common cause
8	I/O ERROR	Door open, or diskette not INITed
9	DISK FULL	Too many files on diskette
10	FILE LOCKED	Attempt to over-write a LOCKed file
11	SYNTAX ERROR	Bad file name, parameter, or comma
12	NO BUFFERS AVAILABLE	Too many text files OPEN
13	FILE TYPE MISMATCH	Diskette file doesn't match command
14	PROGRAM TOO LARGE	Insufficient Apple memory available
15	NOT DIRECT COMMAND	Command must be in a program

DISCUSSION

LANGUAGE NOT AVAILABLE (ONERR GOTO code = 1)

Occurs if DOS cannot find a programming language, either Integer BASIC or Applesoft, that is required to execute a DOS command. The commands FP, INT, LOAD and RUN may all initiate a language search. If Integer BASIC is requested, DOS looks for that language in ROM. If Applesoft is requested, DOS first looks for the language in ROM, using Applesoft from an Applesoft firmware ROM card (if available) regardless of the card's switch position. If Applesoft is not found in ROM, DOS looks on the diskette in the "default" disk drive -- the drive indicated by the default or most recent values of the S and D parameters. DOS will not look on any other disk drive.

This message usually arises after a DOS request for diskette Applesoft, if the diskette in the default drive does not contain the program APPLESOFT. Replace the diskette with one that contains the program APPLESOFT; or use the D parameter with any DOS command, to select the another drive. A command such as this will do nicely:
FP, D2

If you think DOS should have found Integer BASIC in ROM, but it didn't, try the following:

1. Turn off your Apple and remove the cover.
2. Locate the row of four large ROM chips (black, rectangular objects) in the middle of the main printed-circuit board. These chips are labeled "ROM F8", "ROM F0", "ROM E8" and "ROM E0".
3. Press down firmly on these chips.
4. Replace the cover, turn on the Apple and try INT Again.

If you think DOS should have found Applesoft on your firmware ROM card, but it didn't, try the following:

1. Turn off your Apple and remove the cover.
2. Unplug the Applesoft firmware ROM card. Locate the row of five large ROM chips (black, rectangular objects) across the card. These chips are labeled 1, 2, 3, 4, and 5 above the chips, and D0, D8, E0, E8 and F0 below the chips.
3. Press these chips firmly into their sockets.
4. Plug the Applesoft card back into slot #0, the leftmost slot.
5. Replace the cover, turn on the Apple and try FP again.

RANGE ERROR (ONERR GOTO code = 2 or 3)

Occurs when the value of a DOS command parameter or a DOS command quantity is too large or too small. Refer to the manual to see which DOS commands are used with which parameters.

	Parameter	Letter	Range Minimum	Range Maximum
All Files:	Slot	S	1	7
	Drive	D	1	2
	Volume	V	Ø *	254
Sequential Text Files:	Byte	B	Ø	32767
	Relative Field	R	Ø	32767
	Absolute Field (EXEC)	R	Ø	32767
Random-Access Text Files:	Record Length	L	1	32767
	Record Number	R	Ø	32767
Binary Files:	Starting Address	A	Ø	65535
	Number of Bytes	L	1	32767

DOS Command	Quantity	Range Minimum	Range Maximum
PR# s	s	Ø	16 **
IN# s	s	Ø	16 **
MAXFILES n	n	1	16

* Minimum volume number INIT will actually assign to a diskette is 1.

** Maximum slot number built into the Apple II is 7. In deferred-execution mode only, the SYNTAX ERROR message is given for s values from 8 through 16.

Note: The use of values outside the above ranges does not always cause the RANGE ERROR message. Any DOS command parameter or command quantity that is less than Ø or greater than 65535 will cause the SYNTAX ERROR message, not the RANGE ERROR message.

WRITE PROTECTED (ONERR GOTO code = 4)

Occurs when DOS attempts to store information on a diskette, but the disk drive does not detect a "write-protect" notch or cutout on the left side of the diskette's outer case. The following are the most likely causes:

1. There is an adhesive label placed over the diskette's write-protect cutout, to prevent accidentally over-writing or deleting any information on the diskette. This label may be removed, whereupon DOS will SAVE or BSAVE or WRITE to the diskette.

2. There is no write-protect cutout on the diskette. This is true on the System Master diskette, for maximum protection. While not recommended, it

is possible to carefully cut a notch of exactly the correct size and in exactly the correct place. Use another diskette's write-protect notch for a model.

3. If you receive this message while RUNning the COPY program, and the cause is not either 1 or 2, above, you may have inserted the diskette into the drive incorrectly (in any other situation, DOS gives the I/O ERROR message to signal incorrect diskette insertion). Check the diskette's position in the drive, and re-read Chapter I's discussion on inserting diskettes.

END OF DATA (ONERR GOTO code = 5)

Occurs when you try to retrieve information from a portion of a text file where no information has ever been stored. Any byte beyond the last field in a sequential text file, or beyond the last field of each record in a random-access text file, may contain the value Ø. Zero is the ASCII code for a null character, a "nothing", and any command that causes the retrieval of this character results in the END OF DATA message. Remember that only OPEN automatically sets the position-in-the-file pointer back to the file's beginning. The message usually occurs after an INPUT or a GET command, and can arise in several different ways:

1. Too many successive INPUTs or INPUT with too many variables. Each INPUT or INPUT variable causes one additional, adjacent field to be read into the Apple.

2. Too many successive GETs. Each GET reads one additional, adjacent byte or character into the Apple.

3. The B (for Byte) parameter was too large. In sequential files, this parameter must not specify a byte beyond the last RETURN character in the file. In random-access files, the B parameter should not specify a byte beyond the last RETURN character in the currently selected record. Remember, the first byte in a file or a record is byte Ø.

4. The R (for Relative-field position) parameter in a POSITION command was too large. In sequential files, this parameter must not specify a field beyond the last existing field in the file. In random-access files, POSITION's R parameter should not specify a field beyond the last existing field in the currently selected record.

Remember, the R parameter used with POSITION is not the same as the R parameter used with READ. It specifies a field position in the file, relative to the current file position and forward in the file, only. RØ specifies no change in the current file position. R1 jumps the file position ahead to the first byte following the field that contains the current position.

POSITION scans forward through the contents of the file, byte by byte, looking for the Rp-th RETURN character. If it encounters a Ø byte (the null character) before finding the required RETURN character, the END OF DATA message is given immediately: it is not necessary actually to INPUT or GET the null character.

5. The R (for absolute-field position) parameter in an EXEC command was too large. This parameter may specify the first field beyond the last existing field in a file, but attempting to specify the second field beyond the file's end will cause the END OF DATA message. Remember, R0 specifies the first field in a file.

6. The R (for Record) parameter in a READ command specified a random-access file record in which nothing has yet been stored. Before you can READ from a particular record in a random-access file, you must first WRITE some information into that record.

Remember, READ's R parameter is not the same as the R parameter used by POSITION or EXEC. READ's R parameter specifies an absolute record in a file: R0 is the file's first record, and so on.

DOS uses the OPEN command's L parameter for calculating where the Rr-th record begins, so the OPEN preceding READ must use the same L parameter value as the OPEN that preceded WRITE for that file.

FILE NOT FOUND (ONERR GOTO code = 6)

Occurs when certain DOS commands specify a file name that is not in the CATALOG for the diskette in the selected or default disk drive. Only the commands SAVE, BSAVE, INIT and OPEN can create a file whose name did not previously exist. In addition to these, CLOSE may be used with any valid name. A file name specified by any other DOS command must already exist on the diskette.

This message may arise in various ways:
1. You may have misspelled the file's name, by a typing error or by omitting the comma that separates the file name from a following parameter. Check the CATALOG for the exact spelling of the file's name. Warning: if you have accidentally typed control characters into the name of a file, CATALOG will not display these characters. For help, see "File Names" in Appendix F.

2. The file is on another diskette. Check the CATALOG.

3. The file has been accidentally DELETEd. Check the CATALOG.

4. When you use the INIT command or the UPDATE program on a diskette, you specify a file name which DOS thereafter attempts to RUN each time you boot the system with that diskette in disk drive 1. Unless you write a BASIC program, and save it using the name given to INIT or UPDATE, the FILE NOT FOUND message will be given each time the system is booted with that diskette in drive 1. If you can't remember the name of this "greeting program", just UPDATE the diskette again.

VOLUME MISMATCH (ONERR GOTO code = 7)

Occurs when the Volume (V) parameter used in a DOS command is not the same as the volume number assigned to the diskette in the default or selected

disk drive, when that diskette was INITialized. The volume number of a diskette is shown at the head of each CATALOG display. Unless a DOS command specifies a particular volume, the diskette's volume number is ignored, and no message is given. If a DOS command specifies volume Ø, the diskette's volume number is still ignored. If no volume number is given with INIT, or if volume number Ø is given, the diskette will be initialized with the default volume number 254.

I/O ERROR (ONERR GOTO code = 8)

Occurs after an unsuccessful attempt to store data on a diskette or to retrieve data from a diskette (DOS tries 96 times, then gives up). This message can occur in the following ways:

1. The selected or default drive's door is open. Close the door to the disk drive.

2. No diskette in the selected or default disk drive. Put a diskette into the drive and close the drive door.

3. Diskette in the selected or default disk drive has not been INITialized. INIT the diskette (and UPDATE it to a master diskette, if you wish).

4. Diskette is inserted incorrectly. Check the diskette, and re-read the section in Chapter 1 on inserting diskettes.

5. During execution of a VERIFY command, DOS found the specified file was not stored correctly on the diskette. If the file's information is still in memory, try storing it again (perhaps on a different diskette).

6. The DOS command's D (Drive) parameter has specified a disk drive that does not exist in this system. The default drive is now the non-existent drive. Just specify the correct D parameter with the next DOS command to reset the default.

7. The DOS command's S (Slot) parameter has specified a slot that does not contain a disk controller card in this system.

You are in trouble. The default slot is now the empty slot your last DOS command specified. The next DOS command without a slot parameter will go to the empty slot and return the same message as before. Worse yet, DOS thinks the disk drive which does not exist in that slot is still running. The next DOS command specifying the correct slot will send the system into permanent limbo, waiting for the non-existent drive to stop running before it turns on the newly-selected drive. You must either re-boot the system (losing any program in memory, of course) or else:

a)	Type	CATALOG, Ss	(where s = correct slot)
b)	Press	the RESET key	(when the system hangs)
c)	Type	3Ø0G	(system is now okay again)

DISK FULL (ONERR GOTO code = 9)

Occurs when DOS attempts to store information on a diskette, and finds that no more storage space is available on that diskette. A maximum of 403 sectors can been filled with user-stored information, as displayed in the CATALOG (if an individual file exceeds 255 sectors, the CATALOG display of its length starts over again at 000). If you receive the DISK FULL message, rest assured that all files are CLOSEd, and that DOS saved for you all it could (leaving you with some portion of your file not on the diskette). If you receive this message while saving a file called STUFF, the first thing you should do is to
DELETE STUFF
and then save your program on another diskette that has more room left.

If you receive the DISK FULL message and then immediately try to SAVE, BSAVE or WRITE any file on the diskette before DELETEing any files, then (are you ready?) the sector length of the eighth entry shown in the CATALOG will be set to 0. Don't despair: despite the odd appearance of the eighth entry's CATALOG display, the file itself is in fine shape. Other odd events may occur as well. To avoid such situations, if you get a DISK FULL message, DELETE some files before trying to save other files.

FILE LOCKED (ONERR GOTO code = 10)

Occurs when you try to SAVE, BSAVE, WRITE or DELETE using a file name that has been LOCKed on the diskette that is in the selected or default drive. Check the CATALOG display: the names of LOCKed files are preceded by an asterisk (*) in the CATALOG display. A file is LOCKed to prevent accidental over-writing. Use another diskette or UNLOCK the desired file.

SYNTAX ERROR (ONERR GOTO code = 11)

Occurs when DOS encounters a syntax error in a DOS command. Check the manual for the exact syntax required for the command in question. The problem may be a non-valid file name (see Appendix F), an incorrect parameter symbol, a missing parameter, a missing or incorrect separator (usually a comma). This message will also arise if a parameter value or command quantity is a negative number or is greater than 65535, or, in the case of the IN# and PR# commands used in deferred-execution mode if the specified slot is from 8 through 16.

Rarely, every DOS command causes the Applesoft or Integer BASIC Syntax Error message. This usually means that DOS has not been booted or has become "disconnected" from input and output. Try pressing the RESET key, then typing 3D0G to reconnect DOS; or, re-boot the disk.

NO BUFFERS AVAILABLE (ONERR GOTO code = 12)

Occurs when a DOS command requires another file buffer for input or output, and all the available file buffers are already in use. On booting the system, DOS reserves enough space in the Apple's memory for three input-and-output file buffers. A subsequent MAXFILES command can increase or decrease the number of available file buffers, and a CLOSE command can release file buffers currently in use for text files.

Many DOS commands use one file buffer for input or output during their execution, and then relinquish that buffer when execution of the command has ceased.

When a text file is OPENed, a file buffer is assigned to that file for input and output. This buffer remains in use, generally, until its file is CLOSEd either specifically by file name or by the nameless CLOSE that de-allocates all the text-file buffers. A text file is not automatically CLOSEd by a program's coming to an end. To conserve buffer space, CLOSE files as soon as you are through using them. Remember that the next OPEN will re-set the position-in-the-file pointer to the file's beginning.

The MAXFILES command can be used to increase buffer space before writing the program or loading the program into memory. Increasing MAXFILES moves HIMEM down, and this can erase stored Integer BASIC program lines or Applesoft strings. Changing MAXFILES in the middle of a program can be especially dangerous.

FILE TYPE MISMATCH (ONERR GOTO code = 13)

Occurs when a DOS command attempts to use a file name that is already assigned to a file whose file type is inappropriate to the present command. If you are sure the command is correct, use a file name that is not now on the diskette, use a different diskette, RENAME the existing file or DELETE the existing file.

This message arises from several different incorrect combinations of DOS commands with existing file types. Here are the correct combinations:

LOAD f, RUN f, SAVE f	f must be an Applesoft or Integer BASIC program file.
CHAIN f	f must be an Integer BASIC program file.
OPEN f, READ f, WRITE f, APPEND f, POSITION i, EXEC f	f must be a text file.
BLOAD f, BRUN f, BSAVE f	f must be a binary program or data file.

The greeting program's file name, specified with INIT or UPDATE, must refer to an Applesoft or Integer BASIC program file.

PROGRAM TOO LARGE (ONERR GOTO code = 14)

Occurs when a DOS command attempts to place a diskette file into Apple's memory, and finds the available memory insufficient to contain the entire file. You (or a previous program) may have set HIMEM too low for the current task, or a large MAXFILES may have set HIMEM too low. If you set the number of file buffers to three, using the command

MAXFILES 3

then HIMEM will be returned to the booted value given in Appendix D, Table 2.

If you are in Integer BASIC, and HIMEM is set low (to protect the high-resolution screen memory, for instance), you may experience trouble on shifting to diskette Applesoft. Diskette Applesoft occupies about 12.5K of memory, but a shift to diskette Applesoft (with FP or LOAD or RUN) does not reset HIMEM to maximum. When DOS tries to load the Applesoft program from diskette, the message PROGRAM TOO LARGE will be given if HIMEM is below about 13100. The system will be left in Integer BASIC again, and you must set HIMEM higher from Integer BASIC. See Appendix D, Table 2 for your system's maximum HIMEM with DOS and three file buffers.

In deciding whether or not a program will fit into the available memory, DOS looks only at the number of diskette sectors occupied by the program. In general, the program does not completely fill the last sector (256 bytes), but DOS ignores this fact. DOS compares only the high-order byte of LOMEM (Integer BASIC) or HIMEM (Applesoft) with the high-order byte of the projected end-of-program location. Thus a program which should fit into memory, but which would leave less than 256 bytes of free memory after loading, may cause the PROGRAM TOO LARGE message. Sometimes this can be corrected by moving HIMEM or LOMEM slightly, to change the high-order byte, before loading the program.

NOT DIRECT COMMAND (ONERR GOTO code = 15)

Occurs when you try to use one of the text file commands APPEND, OPEN, POSITION, READ or WRITE from immediate-execution mode. These DOS commands can be used only from within PRINT statements in program lines.

APPENDIX C
FORMAT OF DISKETTE INFORMATION

This appendix tells how information is stored on a diskette, and how DOS remembers where particular information has been stored.

In the following discussion, a dollar sign ($) or the label "Hex" preceding a number indicates that the number is expressed in hexadecimal.

OVERVIEW OF THE STORAGE PROCESS

In the Disk II system, information is recorded on a diskette in 35 concentric zones or bands, called tracks. These tracks are numbered from track $00, the outermost, through track $22, the innermost. The disk drive's recording and reading head can be moved in and out, to stop and hover over each of these 35 different zones of the spinning diskette.

Furthermore, the length of each track on the diskette is divided into 13 segments, called sectors. These sectors are numbered from $0 through $C, and up to 256 ($100) bytes of information can be stored in each sector. Once the disk drive's recording and reading head is positioned over a given track, that track's 13 sectors will pass under the head, one after the other, each time the diskette spins around. DOS always records information on the diskette in 256-byte chunks, exactly filling one sector each time.

To store information on the diskette, DOS first puts 256 bytes (one sector's worth) of the information in an area of Apple's memory called a file buffer. When this file buffer is full, the information is stored in one sector on the diskette. Then DOS fills Apple's file buffer with the next 256 bytes of information and stores that information on the diskette.

In general, DOS will begin storing a program or text file wherever it can find an unused sector on the diskette. When that sector is filled with its 256 bytes of information, DOS finds another free sector, perhaps on another track, and continues to record information there. This process continues until the entire file has been stored.

To remember which sectors of which tracks contain the information for a particular file, DOS makes up a list of each track and sector used, as it stores the file. Then DOS stores that list, called a track/sector list, in yet another free sector (or sectors) on the diskette.

Finally, the file's name, file-type, length in sectors, and the diskette location of the file's track/sector list are recorded in a special area of track $11 called the directory. At this time, too, the diskette's track bit map is updated to correctly show which sectors of each track are currently in use.

WRITING INTO A SEQUENTIAL TEXT FILE

Entries in a text file consist of 1 to 32767 characters stored as their equivalent ASCII codes and ended by a RETURN character (either ASCII $8D or ASCII $8D). Each such entry is called a field.

In a sequential text file (no length parameter specified when the file was OPENed), fields are stored immediately following each other (see Chapter 6). DOS writes the first byte of each new field immediately following the RETURN character that ended the previous field (unless otherwise instructed by a Byte parameter). Each time the file is OPENed, DOS forgets the current position within the file, and starts WRITEing again in byte Ø (again, unless otherwise instructed by a Byte parameter).

In order to re-write a particular field or character within a sequential file, WRITE can be used with the B (for Byte) parameter to begin writing at the specified, absolute byte of the file (the first byte in the file is byte Ø, the next is byte 1, etc.). The byte specified may be before or after the current position in the file.

It is very difficult to remember exactly which character appears in every byte of a text file, especially in a sequential text file. For this reason, use of the Byte parameter in sequential text files is not recommended.

The POSITION command can be used with an R (for Relative-field) parameter to move a pointer ahead (only) through the file a specified number of fields relative to the current position in the file. A program portion such as

```
120  PRINT D$;"OPEN NAMES"
130  PRINT D$;"POSITION NAMES, R1
     3"
140  PRINT D$;"WRITE NAMES"
160  PRINT "APPLE COMPUTER"
170  PRINT D$;"CLOSE NAMES"
```

will attempt to WRITE the characters APPLE COMPUTER into the NAMES file, beginning in the first byte of the fourteenth field (the first field is Relative-field Ø).

POSITION can move you to the first byte of any given field Relative to the current position in a sequential text file. If you then re-WRITE that field, however, you must make sure that you re-PRINT exactly the same number of characters that you PRINTed in that field originally. If you PRINT fewer characters, you will have created two new fields: the field you just PRINTed, and the tail-end of the original field you were over-writing. If you PRINT more characters than the original field contained, you will have over-written some of the characters at the start of the next field.

WRITING INTO A RANDOM-ACCESS TEXT FILE

For a random-access text file, a Length parameter is specified when the file is OPENed. The Length parameter determines the number of bytes in a record, which is a field or a collection of fields that DOS treats as a unit. Each record in a random-access text file is like a separate sequential text file whose maximum total length has been specified by the Length parameter. As long as you stay within that maximum Length, you can WRITE and re-WRITE all you want, without affecting any other record in the file. WRITE can be used with the R (for Record) and B (for Byte) parameters to begin writing into any byte of a specified record.

Since any DOS command will terminate WRITE-ing, you cannot use POSITION to jump ahead into different fields within the record specified by the WRITE command.

DOS uses the Length parameter to calculate where to write the first byte of each new record (L bytes beyond the first byte of the previous record). DOS simply skips over any bytes between the previous record's last character and byte L. The bytes skipped over will continue to contain whatever values were stored there at some earlier time (see the next section for details).

If you attempt to WRITE more characters in a random-access record than you specified in the Length parameter, all the characters are stored correctly on the diskette. However, when you begin WRITEing to the next record, DOS calculates the new record's starting position as if the previous record had been within the specified Length. The new record thus overwrites the last characters of the previous, over-sized record, including the end-marking RETURN character of the previous record's last field. The result is messy.

HOW DOS WRITES INTO TEXT FILES: GENERAL PROCEDURE

When you WRITE a field into a text file, DOS first checks on the diskette to see whether or not you have already stored information in the sector which should contain that field. If your file has never used that sector before, DOS places zeros in all 256 bytes of an Apple file buffer, and then lets you put your information into that buffer for later storage in the correct diskette sector. The contents of the file buffer are stored on the diskette when your information has completely filled 256 bytes of the buffer, or when the file is CLOSEd.

Thus, when you WRITE to a particular sector the first time, unused bytes are given the value zero. An attempt to READ a byte containing a zero (the ASCII code for the null character) will result in the message END OF DATA

But if DOS finds your file has already stored information in the sector which should contain the field that you are now WRITEing, it reads all 256 bytes from that sector into the Apple's file buffer. After you have changed any of those file-buffer bytes to contain your new information -- the WRITE, POSITION (sequential files only) and PRINT commands take care of this for you -- DOS then stores the buffer's contents right back into the the same diskette sector where they originated. The contents of the file buffer are stored back on the diskette when you attempt to change any byte not in the sector that was read into the file buffer, or when the file is CLOSEd.

Thus, if you WRITE more information for a file, and DOS stores that information in a diskette sector already being used by your file, this will not re-write any zeros in unused bytes. Any of those sector bytes which you did not re-write will continue to contain whatever information might have been stored there before your WRITE command. This is true of the unused bytes at the end of a sequential text file, and also true of the unused bytes in each fixed-length record of a random-access text file.

CONTENTS OF FILE SECTORS

Now that you know the general process of recording a file on diskette, we can discuss each element in more detail. The actual information stored, sector by sector, is different for each type of file.

FORMAT OF FILE SECTORS
for different file types

File type	Sector	Byte (Hex)	Contents of byte
BASIC (both types)	1st sector	0	Program length, low byte
		1	" " , high byte
		2 through FF	Tokenized program
	Subsequent sectors	All bytes	Tokenized program
Text	All sectors	All bytes	ASCII representation of text: one byte/character ($00 marks end of file)

File type	Sector	Byte (Hex)	Contents of byte
Binary	1st sector	0	Starting RAM address, low byte
		1	" " " , high byte
		2	Length of RAM image, low byte
		3	" " " " , high byte
		4 through FF	Binary data
	Subsequent sectors	All bytes	Binary data

THE TRACK/SECTOR LIST

As a file is stored on the diskette, DOS makes a list of the diskette locations used by the file. This track/sector list is then stored on the diskette in the same way the file itself was stored. The contents of a track/sector list are as follows:

First Sector of a TRACK/SECTOR LIST

Byte (Hex)	Contents of byte
0	Not used
1	Link: track number where continuation of the track/sector list may be found.
2	Link: sector number where continuation of the track/sector list may be found. (If both bytes of Link = 0, no link.)
3 through B	Not used
C	Track number of first file sector
D	Sector " " " " "
E	Track number of second file sector
F	Sector " " " " "
10	Track number of third file sector
11	Sector " " " " "
.	.
.	.
.	.
FE	Track number of 122nd file sector
FF	Sector " " " " "

If any track/sector pair is Ø/Ø this indicates an unassigned sector (usually the end of the file, although text files may contain Ø/Ø indicators for many as-yet-unassigned sectors where future bytes or records may be written).

Subsequent sectors of the track/sector list (if the list extends beyond 122 track/sector pairs) are identical to the first sector described above, except that the track/sector pairs refer to subsequent groups of 122 file sectors. Also, Link bytes 1 and 2 will be different for each subsequent sector. Each Link pair gives DOS the diskette location of the next portion of the track/sector list. If both bytes of the Link are Ø, this indicates the final portion of the track/sector list.

With a text file, only the track/sector pairs for those sectors actually containing information appear as non-zero in the track/sector list. DOS calculates the correct position for the track/sector pair within the list, filling unassigned track/sector pairs with zeros. If a complete sector at the beginning of the track/sector list would contain nothing but zeros, that sector is not stored on the diskette.

Thus, if the Length parameter for a random-access file is 128 (two records per sector) and you WRITE only to record number 2700, only two diskette sectors are actually used: one for the contents of record number 2700, and one for the "twelfth" (and only) sector of the track/sector list. The contents of records number Ø through 2683 may someday occupy 1342 sectors; but until those records are written, they do not use any diskette space. The track/sector list giving the locations of the sectors containing records number Ø to 2683 would have occupied eleven additional sectors, and the list position of the track/sector pair for record number 2700 is calculated as if the entire twelve sectors of list were present. However, since nothing has actually been written to any of the sectors that may someday contain the first 2684 records, DOS does not keep the track/sector list for those unused sectors.

THE DISKETTE DIRECTORY

On every INITialized diskette, track $11 is reserved for information concerning the contents of the diskette. This is where DOS stores the directory containing, for each file, the file's name, its file type, the number of sectors occupied by the file (MOD 256), and the diskette location of the file's track/sector list. The CATALOG command causes most of this information to be displayed on the screen. Each sector of a diskette directory is formatted as follows:

One sector of a DISKETTE DIRECTORY

Byte (Hex)	Contents of byte
Ø	Not used
1	Link: Track number where continuation of the directory may be found (normally $11)
2	Link: Sector number where continuation of the directory may be found (If both bytes of Link = Ø, no link.)

Byte (Hex)	Contents of byte	
3 through A	Not used	
B through 26	Directory entry for file 1	(see below)
2E through 50	Directory entry for file 2	
51 through 73	Directory entry for file 3	
74 through 96	Directory entry for file 4	
97 through B9	Directory entry for file 5	
BA through DC	Directory entry for file 6	
DD through FF	Directory entry for file 7	

The file numbers shown for the seven directory entries are arbitrary. When a file is DELETEd, DOS marks the directory entry for that file (see following table). The next time a file is stored, DOS replaces the old marked directory entry with the directory entry for the new file. Thus, while DOS originally fills the directory in the order shown, file DELETEions soon render this order meaningless.

The diskette directory begins in track $11, sector $C. If more space is needed to store additional directory entries, sector $C is linked to sector $B. If still more space is needed, sector $B is linked to sector $A, and so on, through sector $1. This allows the directory to store directory entries for a maximum of 84 different files.

Each directory entry is written in the following format:

DIRECTORY ENTRY FOR ONE FILE

Relative Byte (Hex)	Contents of Byte
0	Track number of the file's track/sector list (Changed to $FF when the file is DELETEd.)
1	Sector number of the file's track/sector list
2	File type (see discussion on the next page)
3 through 20	File name
21	Sector count: the number of diskette sectors (MOD 256) occupied by the file
22	Kool marks normally zero (but changed to the track location of the track/sector list, from relative byte 0, when the file is DELETEd)

A directory entry's relative byte specifies each byte within the entry, although each entry starts at a different actual byte number within the directory sector. To find the absolute sector byte corresponding to a relative byte, add the relative byte to the entry's first absolute sector byte (as listed in the previous table).

Because only one byte is used to store a file's sector count, the maximum directory sector count is 255 ($FF). If a file exceeds 255 sectors, its sector count (as displayed by CATALOG) starts over again at $00. This does not affect use of the file, but may give an erroneous impression of how full the diskette is.

The eight bits of a file's type-designating byte, relative byte number 2 in a file's directory entry (see previous table), are assigned values as follows:

BYTE INDICATING THE FILE TYPE

Bit	CATALOG symbol	File type designated
7	*	File is locked (write protected) if this bit = 1 File is unlocked (not protected) if this bit = 0
6		Expansion type for future use (normally zero)
5		" " " " " " "
4		" " " " " " "
3		" " " " " " "
2	B	Binary file if this bit = 1
1	A	Applesoft BASIC file if this bit = 1
0	I	Integer BASIC file if this bit = 1
	T	Text file if bits 0 through 6 are all zero

The file type is determined by a 1-bit appearing in one of the bits 0 through 6. If bits 0 through 6 are all 0-bits, the file type defaults to a Text file.

The file's type-designating byte can thus take on the following values:

VALUES FOR BYTE INDICATING FILE TYPE

File type	Value of Type byte (Hex) File unlocked	File locked
Text	0	80
Integer	1	81
Applesoft	2	82
Binary	4	84

VOLUME TABLE OF CONTENTS

Sector $Ø of track $11 contains the diskette's Volume Table of Contents, or VTOC. The VTOC stores the following information:

VOLUME TABLE OF CONTENTS (VTOC)
Track $11, Sector $Ø

Byte (Hex)	Value (Hex)	Description
Ø	2	Not used
1	11	Track number of first directory sector
2	ØC	Sector " " " " "
3	2	DOS release number
4	Ø	Not used
5	Ø	" "
6	1 through FE	Diskette volume number (default: $FE)
7 through 26	Ø	Not used
27	7A	Maximum number of track/sector pairs possible in each sector of a track/sector list
28 through 2F	Ø	Not used
3Ø	FF	These four bytes are a mask for the
31	F8	track bit maps (see next 2 pages):
32	ØØ	each 1-bit enables one of the 13
33	ØØ	sectors to be used in every track.
34	23	Number of tracks per diskette
35	ØD	Number of sectors per track
36	ØØ	Number of bytes per sector, low byte
37	Ø1	" " " " " high byte
38 through 3B	Ø	Track Ø bit map (These tracks
3C through 3F	Ø	Track 1 bit map not available
4Ø through 43	Ø	Track 2 bit map to the user)
44 and 45	?	Track 3 bit map
46 and 47	Ø	" " " "

[Continued on next page]

Byte (Hex)	Value (Hex)	Description
48 and 49	?	Track 4 bit map
4A and 4B	Ø	" " " "
.	.	.
.	.	.
.	.	.
78 and 79	?	Track $10 bit map
7A and 7B	Ø	" " " "
7C through 7F	Ø	Track $11 bit map (Directory & VTOC)
80 and 81	?	Track $12 bit map
82 and 83	Ø	" " " "
.	.	.
.	.	.
.	.	.
C0 and C1	?	Track $22 bit map
C2 and C3	Ø	" " " "
C4 through FF	Ø	Not used

TRACK BIT MAP

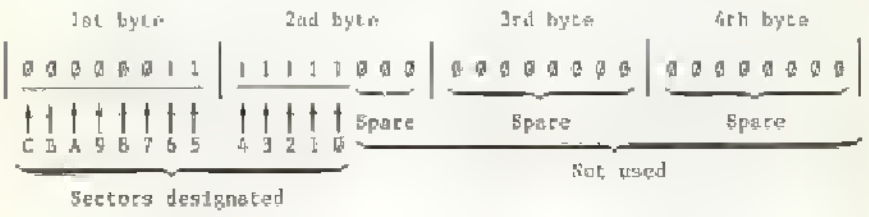
Starting in byte $38 of the VTOC (see previous table), subsequent four-byte groups each contain the track bit map for one of the diskette's 35 tracks. The arrangement of 1-bits and Ø-bits within a track's bit map shows DOS which sectors of that diskette track are currently in use, and which sectors are free. The bit map for each track uses the following format:

TRACK BIT MAP
For one diskette track

Byte	Bit	Designated Sector (Hex)	Byte	Bit	Designated Sector (Hex)
1st	7	C	2nd	7	4
	6	B		6	3
	5	A		5	2
	4	9		4	1
	3	8		3	Ø
	2	7		2 through Ø	Spare
	1	6	- - - - - - - - - - - - - - - -		
	Ø	5	3rd & 4th	All	Spare

If a bit in the track bit map contains the value 1, the sector corresponding to that bit is free. If a bit in the map contains the value Ø, the sector corresponding to that bit is currently in use. Bits marked "Spare" in the table above contain the value Ø; these bits are not used. The track bit map for a typical track might appear as follows:

TYPICAL TRACK BIT MAP



1 = Free sector (assuming the corresponding bit of the mask, VTOC bytes $30 through $33, is also 1)

Ø = Sector in use

When a file is being stored on the diskette (using WRITE, SAVE or BSAVE), an entire track is allocated to the file at once (when possible), and the track's bit map shows the entire track in use. Then, when the file is CLOSEd, those sectors not actually used are again designated as free, in the bit map for that track.

Sectors actually used to store a file's information, however, can only be "set free" when that file name is DELETEd. Suppose your diskette contains a 100-sector BASIC file named BIG, for instance. If you now SAVE, on the same diskette, a 2-sector file with the same name BIG (overwriting the old file) a CATALOG of the diskette will reveal that your 2-sector file BIG is still using up 100 sectors. To free up unnecessary sectors used by a BASIC file named BIG, use the following sequence of commands:

```
LOAD BIG
DELETE BIG
SAVE BIG
```

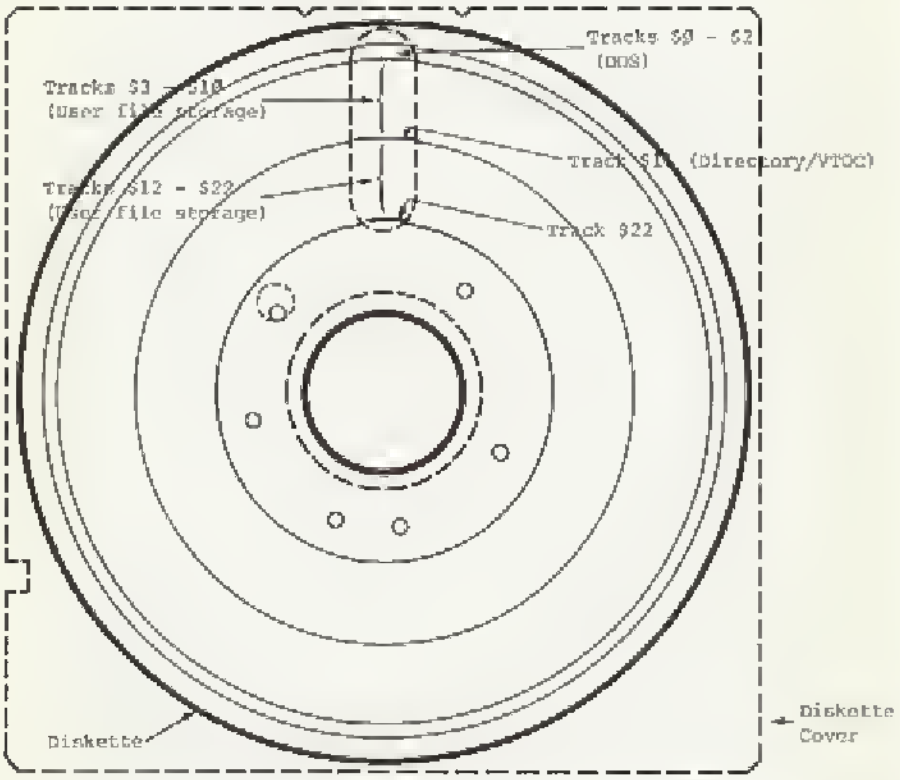
A similar process can be used to release unnecessary sectors used by binary files.

To release unnecessary sectors being used by a text file, you will have to READ each of the file's fields into the Apple. If you store all the fields in an array, you can then DELETE the original file before WRITEing each record back onto the diskette using the original file name. Another way to do this is to read each field into the Apple and immediately WRITE the field back onto the diskette using a file name that is different from the original file name. When you have read and re-written the last field, you can DELETE the original file.

TRACK AND SECTOR ALLOCATION ORDER

Each diskette contains 35 tracks, three of which are reserved for DOS and one for the Directory, leaving 31 tracks for the user. Each track contains 13 sectors, so all together 31*13 or 403 sectors are available to the user.

Sectors are filled starting with sector $C and working back to sector $0. Tracks are first filled starting with track $12 (just inside the directory/VTOC track) and proceeding inward to track $22 (the innermost track). When track $22 has been filled, tracks are then filled starting with track $10 (just outside the directory/VTOC track) and working outward to track $3 (the outermost track available to the user).



TRACK ALLOCATION ORDER

	First Filled		Last Filled
First:	$12	--->	$22
Then:	$10	--->	$03

SECTOR ALLOCATION ORDER

First Filled		Last Filled
$C	--->	$0

RETRIEVING INFORMATION FROM THE DISK

To retrieve a file from diskette, DOS follows the process used to store the file, but in reverse. After a command such as

```
LOAD FILE
```

or

```
BLOAD FILE
```

for instance, DOS goes to the diskette's file directory in track $11, and finds the directory entry containing the name FILE. This entry also contains the diskette location (by track and sector) of the desired file's track/sector list. DOS then goes to this track/sector list, and reads the first track/sector pair. This pair specifies the diskette location of the first sector containing the program named FILE. When DOS has read that first sector of program into the Apple, it returns to the track/sector list for the location of the program's second sector, and so on.

READING FROM A SEQUENTIAL FILE

When READing from a sequential text file, with a program portion such as

```
50 PRINT D$; "READ TEXTFILE"
60 INPUT A$
```

for instance, the general process is like that described for LOADing a program file. However, only the sector containing the text file's next field (all characters from the current position in the file through the next RETURN character) is read into the Apple's file buffer in response to the INPUT command. Then the actual sector bytes that make up the desired field are assigned to the variable A$. This process is repeated if the field extends over more than one diskette sector. Each subsequent INPUT command will cause reading of the file to resume, from the Apple's file buffer if it already contains the proper field, or by reading another diskette sector into the Apple. This continues until the last field is read or some command CLOSEs the file.

By using the READ command with the B (for Byte) parameter, you can cause the next INPUT to begin reading from the specified absolute byte in the file (the file's first byte is 0, the next is 1, etc.). This byte may be before or after the current position within the file. To use this parameter effectively, however, you must know the contents of every byte in your file. The POSITION command uses the R (for Relative-field) parameter to move DOS's current-position pointer the specified number of fields forward (only) through the file, relative to the current position in the file. Each time you OPEN a file, DOS forgets its current position in the file and starts READing again from the beginning of the file (unless otherwise instructed by a Byte parameter).

The INPUT command treats a response somewhat differently in Integer BASIC and in Applesoft. If certain characters such as the colon or comma appear in the response field, further characters in the field may be ignored or assigned to multiple INPUT variables (if any). For details, see the appropriate manual for Integer BASIC or for Applesoft.

READING FROM A RANDOM-ACCESS FILE

The text-reading process is somewhat different when READing from a specified record of a random-access text file (also see WRITING TO A RANDOM-ACCESS FILE in this appendix). In a random-access text file, each record is composed of the same number of bytes, specified in the Length parameter when the file was OPENed prior to WRITEing the file. When this same file is OPENed prior to READing it, an identical Length parameter is given. To find the beginning of a particular record (specified by the READ command's R parameter), DOS uses the Length parameter to calculate the number of bytes occupied by all the preceding records. That number is then divided by 256 ($100) to determine how many file sectors DOS must skip over to reach the sector containing the desired record. Then DOS examines the file's track/sector list and finds the diskette location of the desired file sector. Finally, DOS reads the correct sector into the Apple's file buffer. Then the correct bytes can be read from the file buffer.

This same retrieval process would be followed even if the text file had originally been stored as a sequential file, or as a random-access file using a completely different Length. DOS blindly calculates the sector and byte position of the requested record according to whatever Length parameter you specify when you OPEN the file prior to READing from it, regardless of the Length parameter (if any) that was used when WRITEing the file in the first place.

By using the READ command with both R (for Record) and B (for Byte) parameters, you can cause the next INPUT to begin reading from the specified absolute byte in the specified record (each record's first byte is #0, the next is 1, etc.). This byte may be before or after the current position within the record. To use this parameter effectively, however, you must know the contents of every byte in the specified record.

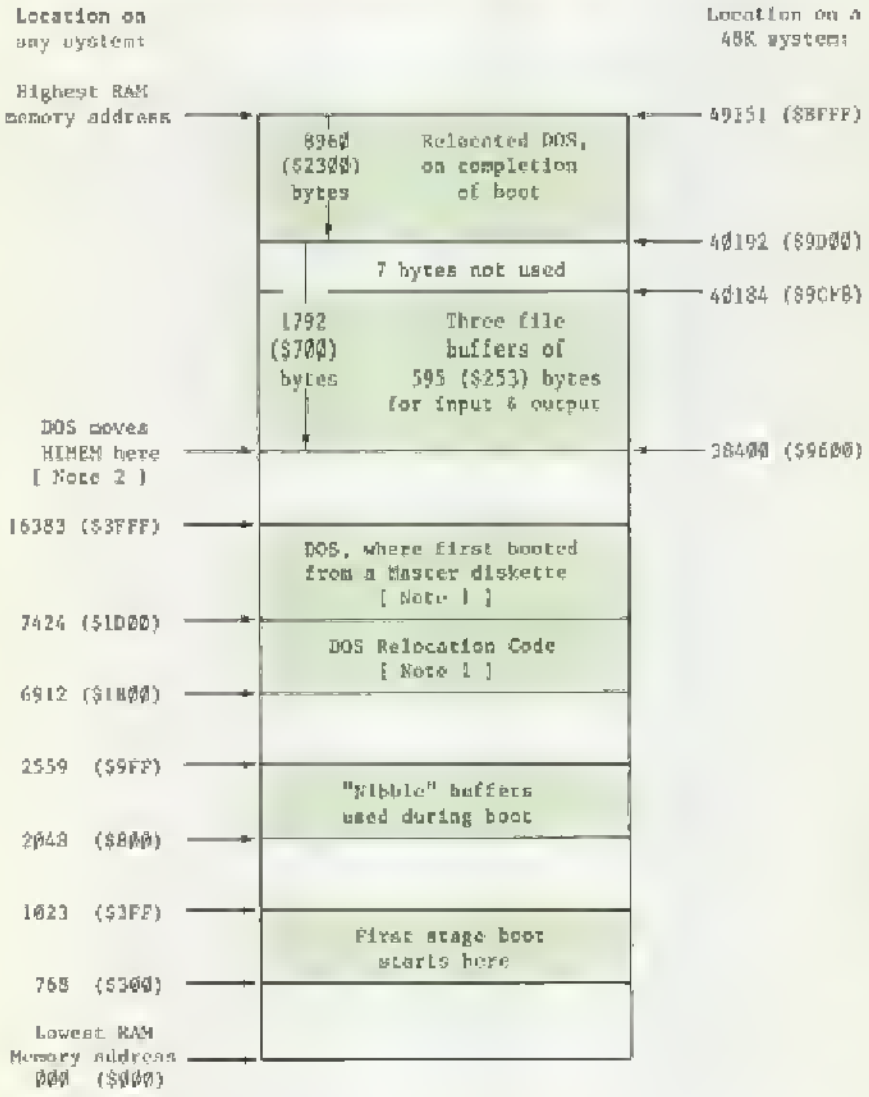
The POSITION command, while primarily intended for access to sequential files, can be used with the R (for Relative-field) parameter to move DOS's current-position pointer the specified number of fields forward (only) through the current record, relative to the current position in the record. READ is used with the R (for Record, this time) parameter to move the current-position pointer to the beginning of the desired record. Using POSITION cancels READ mode (without resetting the position-pointer), and another READ (this time, with no parameter) re-instates READ mode.

Each time you OPEN a file, DOS forgets its current position in the file and starts READing again from the beginning of the file (unless otherwise instructed by a Byte and/or Record parameter).

DOS keeps no information for you concerning the structure, format, record-length, or field-length of your text files. To use your random-access text files effectively, you must keep detailed written information about the structure of these files, or keep the information at the beginning of the file..

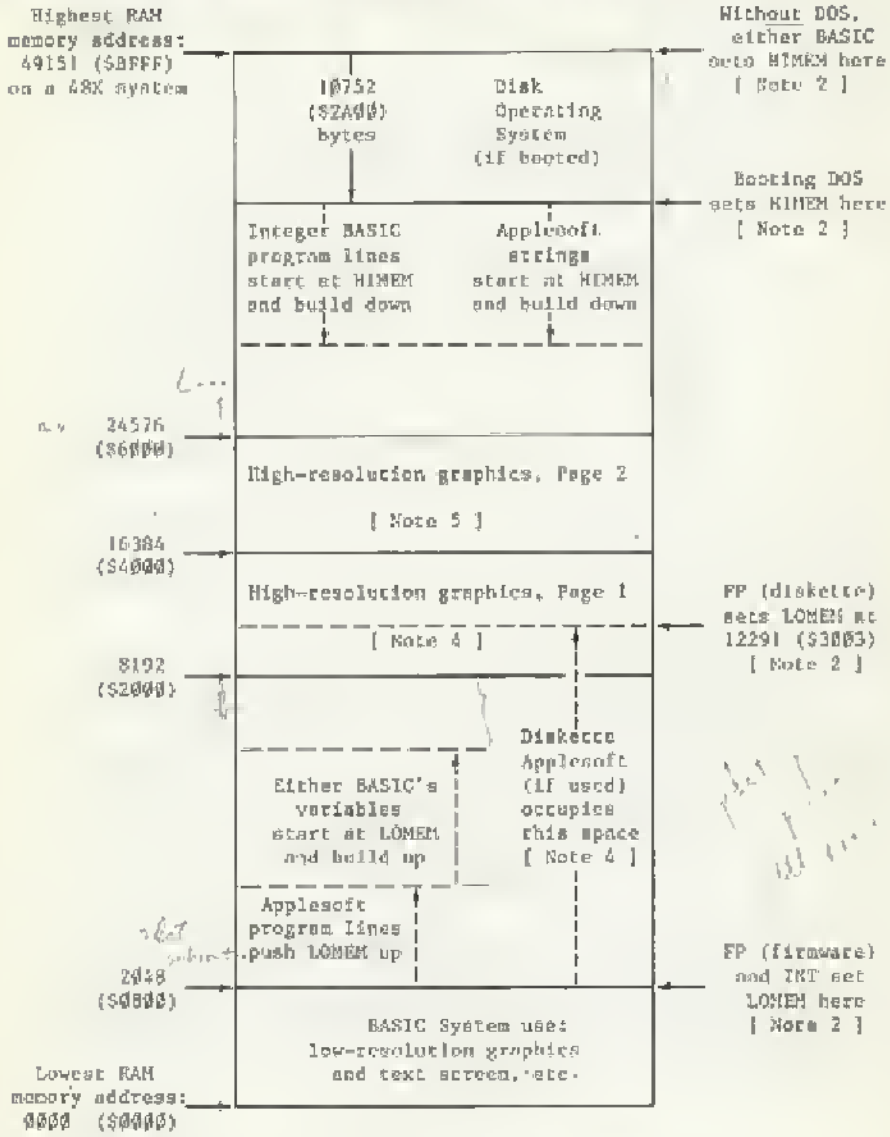
APPENDIX D
MEMORY USAGE

TABLE 1: APPLE II MEMORY MAPS
A. MEMORY AREAS OVER-WRITTEN WHEN BOOTING DOS



Note 1. This memory area is not affected when booting a Slave diskette: DOS is placed directly below the Highest RAM Memory address that was available on the system that INITialized the Slave diskette, whether appropriate to the present system or not.

B. MEMORY AREAS USED BY DOS AND EITHER BASIC



Note 2. If your system is in Integer BASIC, the HIMEM pointer can be found (low byte first, then high byte) in locations 76-77 ($4C-54D). If your system is in APPLESOFT BASIC, the HIMEM pointer is in locations

115-116 ($73-$74), same format. See Table 2 for the value of HIMEM set by booting DOS. Increasing MAXFILES will move HIMEM down an additional 595 bytes for each file buffer added. For the locations of other Applesoft program pointers, consult your Applesoft II BASIC Programming Manual, Appendix I.

TABLE 2: HIMEM VALUE SET BY BOOTING DOS

When DOS is booted, HIMEM is set according to the amount of memory in the system:

System size	Highest RAM address Decimal	Hexadecimal	HIMEM: set by DOS boot Decimal	Hexadecimal	
16K	16383	$3FFF	5632	$1600	
20K	20479	$4FFF	9728	$2600	
24K	24575	$5FFF	13824	$3600	
32K	32767	$7FFF	22016	$5600	
36K	36863	$8FFF	26112	$6600	
48K	49151	$BFFF	-27136	$9600	[Note 3]

Note 3. The number -27136 could also be written 38400, but Integer BASIC will not accept numbers greater than 32767. In Integer BASIC, memory addresses greater than 32767 must be expressed as their negative equivalents. The negative equivalent of any positive decimal address n is (n - 65536).

Note 4. Using high-resolution graphics Page 1 erases the contents of memory locations 8192 through 16383. Unless DOS sets HIMEM to a value greater than 16383, an attempt to use high-resolution graphics Page 1 will erase part of DOS. This means that you cannot use Disk II and high-resolution graphics at the same time, unless your system contains at least 32K of memory.

If you are using diskette Applesoft, an attempt to use high-resolution graphics Page 1 will erase part of Applesoft. With diskette Applesoft, you may use high-resolution graphics Page 2 only, if your system contains at least 36K of memory. See Note 5.

Note 5. Using high-resolution graphics Page 2 erases the contents of memory locations 16384 through 24575. Unless DOS sets HIMEM to a value greater than 24575, an attempt to use high-resolution graphics Page 2 may erase part of DOS. This means that you cannot use Disk II and Page 2 high-resolution graphics at the same time, unless your system contains at least 36K of memory.

APPENDIX E
DOS ENTRY POINTS AND SCHEMATICS

DOS ENTRY POINTS

Routine to re-connect DOS (if page 3 is over-written):

System size	Decimal address (CALL)	Hexadecimal address (C)
48K	-25153	$9D8F
32K	23999	$5D8F
16K	7615	$1D8F

The Monitor command 3D0L displays this number at the top right.

Locations containing the start address and length of a BLOADed program:

System size	Start address (low byte) Decimal	Hexadecimal	Program length (low byte) Decimal	Hexadecimal
48K	43634	$AA72	43616	$AA60
32K	27250	$6A72	27232	$6A60
16K	10866	$2A72	10848	$2A60

To see the starting address or length after a BLOAD, type
PRINT PEEK(low byte) + PEEK(low byte +1)*256

Program to find the DOS locations containing the starting address and length of the most recently BLOADed program, on any size system:

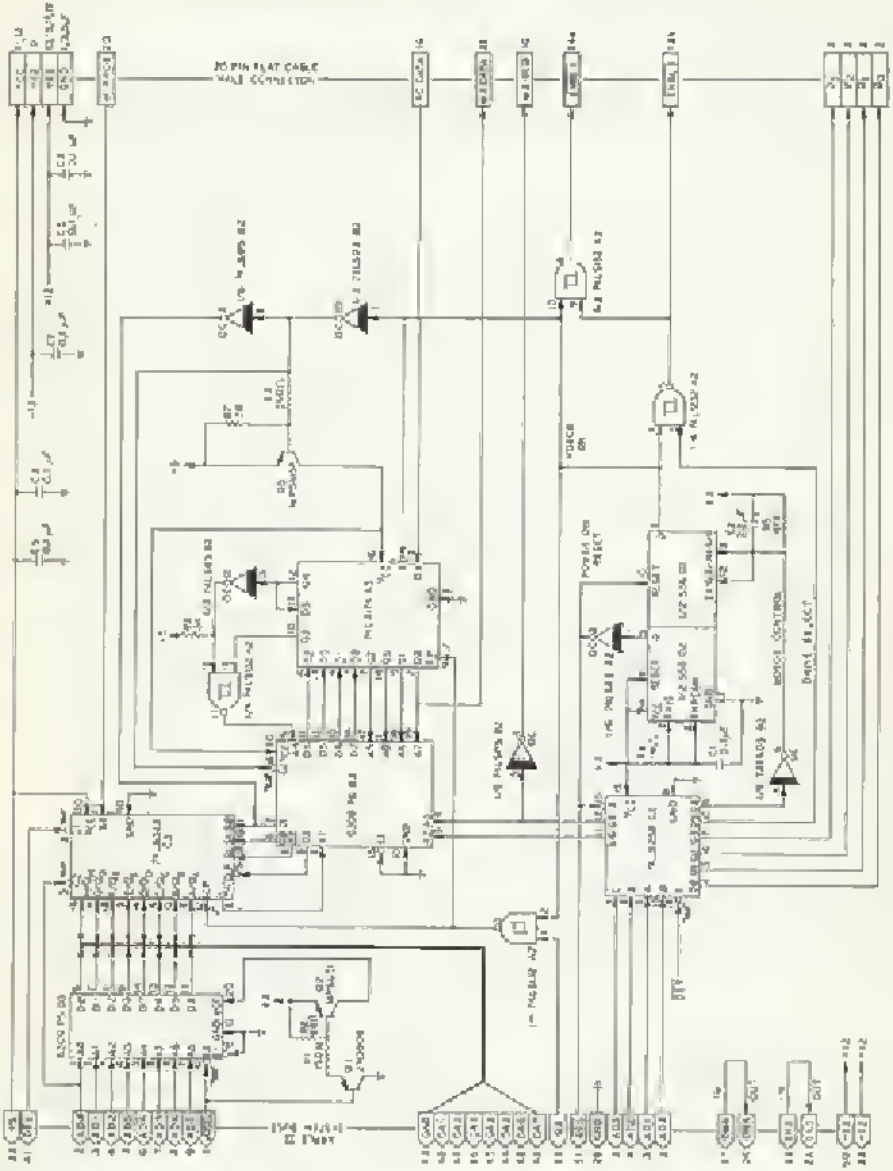
```
5  REM  BLOAD FINDER
7  H = 256 * PEEK (104) - 1: REM HIGHEST ADDRESS
8  T = 43152 - H: REM LOWEST ADDRESS
10 D$ = CHR$ (4): REM CTRL-D
20 PRINT D$;"OPEN FOO,D7777": CLOSE
30 PRINT D$;"BLOAD FOO"
40 PRINT D$;"DELETE FOO"
50 FOR I = H + 1 TO T
60 IF PEEK (I) = 119 OR PEEK
   (I + 1) < > 119 THEN NEXT I
70 PRINT "LOCATIONS OF START
   ADDRESS ="; I; " "; I + 1
80 FOR I = H + 1 TO T
90 IF PEEK (I) < > 119 OR PEEK
   (I + 1) = 0 THEN NEXT I
100 PRINT "LOCATIONS OF LENGTH ="
    I; ","; I + 1
```

The values of H and T (lines 7 and 8) are shown for a 48K system. Appendix D, page 142, shows the correct values for your system. This program takes about 2 minutes to find the desired locations.

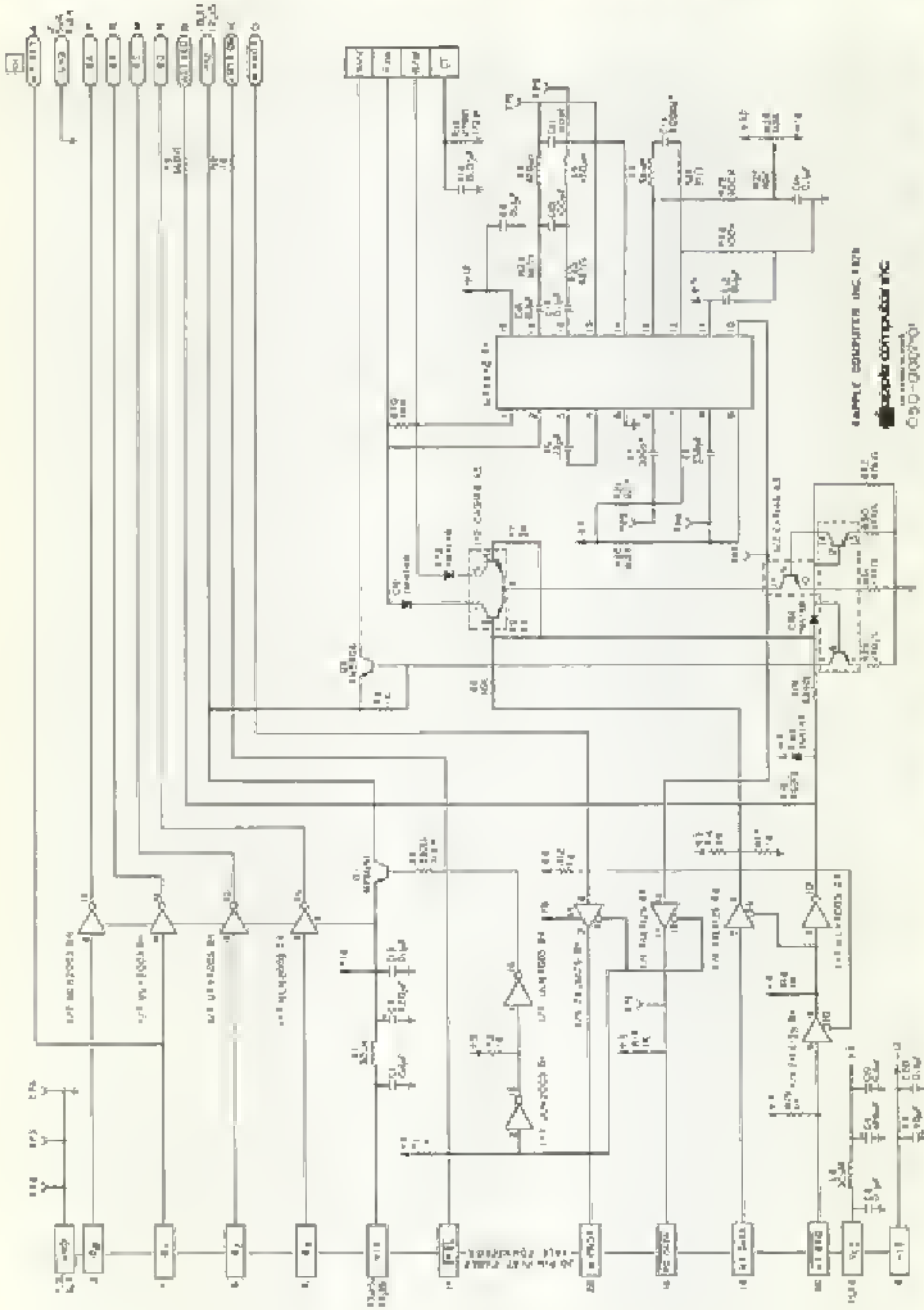
DOS character input and output routines:

See Chapter 10, especially Note 7 on page 105. For an example using the technique described, see the program on page 151.

CIRCUIT SCHEMATIC: DISK II INTERFACE



CIRCUIT SCHEMATIC : DISK II ANALOG BOARD



APPENDIX F
SUMMARY OF DOS COMMANDS

The DOS commands are grouped into 5 categories in this appendix:

Housekeeping commands

INIT	RENAME	VERIFY
CATALOG	DELETE	MON
SAVE	LOCK	NOMON
LOAD	UNLOCK	MAXFILES
RUN		

Access Commands

FP	PR#	CHAIN
INT	IN#	

Sequential Text File Commands

OPEN	APPEND
CLOSE	POSITION
READ	EXEC
WRITE	

Random-Access Text File Commands

OPEN	READ
CLOSE	WRITE

Machine Language File Commands

BSAVE
BLOAD
BRUN

Procedures used in DOS (including chaining in Applesoft) are summarized in Appendix G. The notation used in the summaries (and throughout the manual) is described in the following section.

NOTATION

Syntax refers to the structure of a computer command. A simple notation is used to describe the syntax of each DOS command.

Items in square brackets, [and] , are optional. These items are sometimes called parameters. Not all commands permit all parameters, but those parameters that are permitted in a given command may appear in any order, unless otherwise noted.

If a command uses a file name, the file name must come immediately after the command word itself: the first item following the command will be treated as a file name. The file name must be separated by a comma from any parameter that follows.

Curly brackets may be used to indicate when a certain key should be pressed:

{CTRL} hold down the key marked "CTRL" while another key is typed. {CTRL}D means hold down the CTRL key while you type the letter D. Sometimes another notation is used: CTRL-D means the same as {CTRL}D.

{RETURN} press the key marked "RETURN". The {RETURN} required after every command is not shown.

{RESET} press the key marked "RESET".

{ESC} press the key marked "ESC".

CAPITAL letters and commas must be typed as shown, lower case letters stand for items that you must supply.

f file name. This is from one to 30 characters. Any typeable character except the comma may appear in a file name. The first character must be a letter of the alphabet. For more details, see the next section.
Examples: CHESS
RECIPE
SUM OF SQUARES
R3W45
HOW-ABOUT-THIS

g another file name.
Example: SEPARATOR WITH LOW VELOCITY

s slot number. s specifies the Apple II slot in which the disk controller card has been placed (usually slot 6). s initially defaults to the slot from which DOS was booted. It subsequently defaults to the last value specified for this parameter. s must be in the range 1 through 7.
Examples: 7
2

If s refers to a slot which does not contain a disk controller card, the system may stop and a program in memory may even be lost. See I/O ERROR, in Appendix B, for more details.

v volume number of a diskette. v initially defaults to the volume number of the diskette from which the system was booted. It subsequently defaults to the latest value specified for this parameter, or implicitly specified by a CATALOG command. v must be in the range 0 through 254.
Example: 101
Note: A diskette's volume number may not be 0. In a DOS command, specifying a volume number of 0 or simply V with no number is a "wild card" and tells the DOS to determine and use the volume number on the diskette.

d drive number (either 1 or 2). d initially defaults to one. It subsequently defaults to the latest value specified for this parameter.
Example: 2

p position number. Used with the R parameter in the POSITION and EXEC commands for sequential text files. p specifies a field whose position in the file is p fields ahead of the current file position. p defaults to Ø, which does not move the file-position pointer in the file. Note: EXEC always sets the pointer to the start of the named file, so p is always relative to Ø when used with EXEC. See command summaries later in this Appendix. p must be in the range Ø through 32767.

r record number. Used with the R parameter in the READ and WRITE commands for random-access text files. r defaults to Ø after OPEN. Thereafter, it defaults to the last record specified. r points to an absolute record within a random-access file. r must be in the range Ø through 32767.

a address in RAM. The a parameter is required with the BSAVE command. a specifies the starting Apple memory address for BSAVEing or BLOADing binary information. If BLOAD does not specify an a parameter, then the value of a defaults to that used when the binary file was BSAVEd. a must be in the range Ø through 65535.

b byte number. b defaults to Ø. In a sequential file, b points to an absolute byte within the file. In a random-access file, b points to an absolute byte within the record pointed to by r . b must be in the range Ø to 32767. For most applications b is in the range Ø through the last byte in the current sequential file or the last byte in the current random-access record.

j length specifier. j defaults to 1. When used in the OPEN command with random-access files, j is required and specifies the number of bytes that will constitute a record in a random-access file. When used with the BSAVE command, j is required and specifies the number of bytes of Apple memory, starting at address a , whose contents are to be stored on diskette. j must be in the range Ø through 32767.

As an example of this notation, the DOS command that is notated
INIT f [,Vv] [,Ss] [,Dd]
can be interpreted as
INIT HELLO, V17, D2
by the following process. The keyword "INIT" is in upper case, and must be typed exactly as shown. In the syntax description, "f" is lower case and stands for a file name -- it is replaced by the legitimate file name "HELLO" in this example. The ",V17" is optional. "V" stands for "volume"; 17 was chosen arbitrarily as a volume number for this example. The notation ",Ss" is optional and omitted. The notation ",Dd" becomes ,D2 in this example, indicating that disk drive number 2 is to be used.

Any numerical constant in a DOS command can be entered in hexadecimal notation by preceeding the hexadecimal digits with a dollar sign.

FILE NAMES

File names may be up to 30 characters long, and must begin with a letter. The name cannot contain a comma, a CTRL-M or a RETURN, which is used to terminate the command. Spaces that precede the first non-space character in a name are ignored. All name characters beyond the 30th are ignored.

When typing file names, the use of special keys such as ESC, the left-arrow and right-arrow keys, and certain keys typed with the CTRL key ("control" characters CTRL-C, CTRL-H) may have unexpected effects.

If a file name contains control characters, you won't see them printed, but they must be typed to use or delete the file.

The following Applesoft program can be used to find any hidden characters except CTRL-M (RETURN), ESC, CTRL-H (left arrow) and CTRL-U (right arrow).

```
10  DATA 201,141,240,21,201,136
20  DIM J%(17),I% 201 136 144,1
30  DATA 201,160,176,3,72,152
40  DATA 5,56,223,64,76,249
50  DATA 237,76,240,253
60  FOR I    168 TO 168 + 27
70  READ V  POKE I,V: NEXT I
80  POKE 54,0  POKE 55,3
90  CALL 1000
```

If you suspect you may have accidentally introduced a control character into a file name, type this program, SAVE it, and RUN it. The Applesoft prompt ( ]) will be displayed. Next type

```
CATALOG
```

and you'll get a list of all the files, with any control characters shown as flashing characters. Control characters in program listings can also be found this way. To re-instate normal printouts, type

```
PR# 0
```

HOUSEKEEPING COMMANDS

```
INIT f [,Vv] [,Ss] [,Dd]
```

Example: INIT HELLO, V18

The parameter v assigns a volume number to the diskette being initialized. Details on initializing diskettes are in Chapter 2 and Appendix G.

CATALOG [,Ss] [,Dd]

Example: CATALOG

Displays on the screen the volume number and a list of all files on the diskette in the specified or default drive. The default volume number is changed to match that of the indicated diskette. If this command uses a volume parameter [,Vv] that parameter is ignored.

With each file is displayed an indicator of its file type and the number of diskette sectors occupied by the file. The file types are:

I	Integer BASIC program file, created by SAVE.
A	Applesoft BASIC program file, created by SAVE.
T	Text file, created by OPEN and filled by WRITE.
B	Binary memory-image file, created by BSAVE.

An asterisk beside a file's type indicator shows that the file is LOCKed.

A maximum of 403 diskette sectors are available to the user. Each diskette sector can store up to 256 bytes of information. The minimum length of a file is 1 sector, for an empty text file. (Technically, that 1 sector is occupied by the empty track/sector list for the file.) Empty integer BASIC, Applesoft, and Machine Language files take 2 sectors. (1 for the track/sector list and 1 for the first program sector, which contains the program's length. See Appendix C for more details.)

If an individual file exceeds 255 sectors, the CATALOG display of that file's length starts over at 000. This does not affect use of the file, but may give an erroneous impression of how full the diskette is.

SAVE f [,Ss] [,Dd] [,Vv]

Example: SAVE COLOR DEMOS, V56

If there is no file with the specified file name on the diskette in the specified or default drive, a file is created on that diskette and the current Integer BASIC or Applesoft program is stored under the given file name. If the diskette contains a file with the specified file name, but of a different language or file type, then the message
FILE TYPE MISMATCH
will be displayed.

If the chosen diskette already contains a file with the specified file name, and in the same language, the original file's contents are lost and the current BASIC program is saved in its place. No warning is given.

LOAD f [,Ss] [,Dd] [,Vv]

Example: LOAD DON JONES. V19, D1

Attempts to find Integer BASIC or Applesoft program file with name f on the diskette in the specified or default drive. If the volume numbers match and there is such a file, that program will be LOADed into the computer. It can then be LISTed, or RUN, or SAVEd as with any program. LOAD closes any open text files, changes the Apple to the correct language for file f , and erases any program in memory before placing the new program in the Apple.

If file f is an Applesoft BASIC program, and Applesoft is not already in memory or available from the Applesoft firmware ROM card, the program Applesoft will be LOADed and RUN from the specified drive automatically, before file f is LOADed. If Applesoft is not on that diskette nor on the firmware ROM card, the message
LANGUAGE NOT AVAILABLE
will be displayed.

The instruction LOAD, without any parameters, will LOAD a program from cassette tape.

RUN f [,Ss] [,Dd] [,Vv]

Example: RUN ANNUITY, D2

LOADs file f from the specified or default drive (see the previous discussion), then also RUNs the program loaded. If just
RUN
is typed, the program in memory is RUN.

RENAME f, g [,Ss] [,Dd] [,Vv]

Example: RENAME SEPERATE, SEPARATE, S4, D1, V0

Finds the file named f on the diskette in the specified or default drive, and changes its name to g . The file's contents are not affected. If file f was open, it is closed.

RENAME does not check to see whether the file name g is already in use, so it is possible to use RENAME to put several files of the same name onto a diskette -- a potentially confusing situation, at best.

Do not RENAME the greeting program that was created when the disk was INITialized unless you've first changed the name using the UPDATE 3.2 program. Otherwise, DOS will continue to look for the old greeting program name, each time you boot the system with this diskette in drive 1.

DELETE f [,Ss] [,Dd] [,Vv]

Example: DELETE TEST

Removes the file named f from the diskette in the specified or default drive. If f was open, this command closes it. See Appendix G for more details of the deletion process.

If a file named f does not exist on the diskette, the message
FILE NOT FOUND
will result. To avoid this occurrence stopping your programs, first OPEN the file, then DELETE it.

LOCK f [,Ss] [,Dd] [,Vv]

Example: LOCK LOVE LETTERS, V3!

This command allows you to make file f, on the diskette in the specified or default drive, safe from accidental deletion or change. A LOCKed file is indicated in the CATALOG by an asterisk (*).

UNLOCK f [,Ss] [,Dd] [,Vv]

Example: UNLOCK RECIPES, V3!, D2

If you change your mind, and want to alter or remove a LOCKed file named f, on the diskette in the specified or default drive, this command allows such a change.

VERIFY f [,Ss] [,Dd] [,Vv]

Example: VERIFY SAM

Performs a check that the information actually stored on the diskette in file f is self-consistent. (Technically, this is what happens: When the file is created -- with SAVE, BSAVE or WRITE -- DOS calculates a checksum byte for the contents of each output buffer and then stores that byte with the buffer's contents in a diskette sector. The VERIFY command calculates a new checksum byte for the actual contents of each file sector, and compares it with the checksum byte originally stored with that sector.) If a file VERIFYs, no message is given; it's safe to assume the information on the diskette has been stored correctly. If a file does not VERIFY, the message
I/O ERROR
is presented. You may VERIFY any type of file.

MON [C] [,I] [,O]

Examples: MON O
MON C, I, O

All disk commands and all information sent between the computer and the disk are normally not displayed on the screen. This command allows you to enable some or all of this display -- a helpful tool when debugging a program. If C is specified then disk commands are displayed. If I is specified, then information being sent from the disk to the Apple, as Apple's input, will be displayed. If O is specified, then information being sent to the disk from the Apple, as Apple's output, will be displayed.

At least one of the three parameters must be present, or MON is ignored. The parameters may appear in any order, separated by commas. These parameters appear only in the commands MON and NOMON.

Note: MON remains in effect until a NOMON command, a change of language (FP or INT), a boot, or a restart (3D0G). Even RUNning a program won't cancel a MON.

NOMON [C] [,I] [,O]

Examples: NOMON C
NOMON I, C

The MON command enables you to display disk commands and/or information sent between the computer and the disk: such information is not normally displayed on the screen. The NOMON command allows you to disable some or all of this display. The command
NOMON C, I, O
returns the system to its usual, default state.

If C is specified then disk commands are not displayed. If I is specified, then information being sent from the disk to the Apple, as Apple's input, will not be displayed. If O is specified, then information being sent to the disk from the Apple, as Apple's output, will not be displayed.

At least one of the three parameters must be present, or NOMON is ignored. The parameters may appear in any order, separated by commas. These parameters appear only in the commands MON and NOMON.

MAXFILES n

Example: MAXFILES 6

n is an integer from 1 to 16 that specifies the number of files that can be active at one time. When MAXFILES is executed, 595 bytes of memory (called a file buffer) are reserved for each file. When you boot the system, n defaults to 3, so that you will have 1785 bytes reserved for file buffers and will be allowed a maximum of 3 files open simultaneously.

All DOS commands except PR#, IN# and MAXFILES require a file buffer. Thus if you have MAXFILES 1, and one file is OPEN, an attempt to perform a DOS command (such as CATALOG) will cause the message
NO BUFFERS AVAILABLE
to be displayed.

Use of MAXFILES moves HIMEM. This can erase Integer BASIC programs or Applesoft strings. Use MAXFILES before LOADing and RUNning a program. See the discussion in Chapters 5 and 7 if MAXFILES must be used from within a program.

ACCESS COMMANDS

FP [,Ss] [,Dd] [,Vv]

Example: FP, D2

This command puts your system into Applesoft BASIC. Any Integer BASIC or Applesoft program in memory is lost. If your computer contains the Applesoft firmware card, DOS uses that source for the language, regardless of the switch position on the card. If your system does not contain the Applesoft firmware card, DOS attempts to load and run the program named APPLESOFT on the diskette in the specified or default drive.

To place the APPLESOFT program onto a newly initialized diskette, first LOAD the APPLESOFT program from the Master Diskette, then (without RUNning or LISTing the file) SAVE APPLESOFT on an initialized diskette. You must use the name APPLESOFT for this file.

Do not use RUN APPLESOFT to change languages. Everything looks fine at first, but DOS has not properly initialized the language. To avoid the resultant mess, always use FP.

INT

Example: INT

This command puts the Apple into Integer BASIC. Any Integer BASIC or Applesoft program in memory is lost.

CTRL-D (also written (CTRL)D)

Example: 10 D$=CHR$(4)
20 PRINT D$;"WRITE CHESS"

Every character PRINTed out by the Apple is first examined by DOS before it is sent on to the outside world. If the Apple PRINTs out a RETURN character (most PRINT statements automatically end with a RETURN), and the next character is a CTRL-D, this is a message to DOS that subsequent characters (until the next RETURN) are a DOS command. Most DOS commands may be used from inside an Integer BASIC or Applesoft program. To do so, PRINT a string consisting of CTRL-D followed by the desired DOS command.

The recommended way to do this is to first create a string D$ consisting only of a CTRL-D, and then to use BASIC statements such as shown in the example. Note the use of CHR$(4) to create D$ (this works only in Applesoft, since the CHR$ function is not offerred in Integer BASIC). Instead, CTRL-D could have been typed inside quotation marks to create D$, but in this case no character is shown between the quotation marks.

Every character sent out by the Apple is first examined by DOS before it is passed on to the outside world. If the Apple sends out a RETURN character (most PRINT statements automatically end with a RETURN), and the next character is a CTRL-D, this is a signal to DOS that subsequent characters (until the next RETURN) are a DOS command. A DOS command from a program must appear in a PRINT statement whose first ouput character is CTRL-D and whose output is separated from preceding and from succeeding printed output by RETURN's. For additional information, see "Use of DOS from within a Program", in Appendix G.

PR# s

Example: PR# 6

Sends subsequent Apple output to the device controlled from slot # s , instead of to the TV screen. The command
PR# 0
returns output to the TV screen. If the command is used from inside programs, it must appear as a PRINTed DOS command, as shown below:
10 D$=CHR$(4): REM CTRL-D
20 PRINT D$; "PR# 1"
If no device controller card is installed in slot # s , the system may "hang" and you'll have to press the RESET key to recover

IN# s

Example: IN# 5

Takes subsequent Apple input from the device controlled from slot # s , instead of from the Apple keyboard. The command
IN# 0
resets the normal keyboard input. If the command is used from inside programs, it must appear in a PRINTed DOS command, as shown below:
10 D$=CHR$(4): REM CTRL-D
20 PRINT D$; "IN# 1"
If no device controller card is installed in slot # s , the system may "hang" and you'll have to press the RESET key to recover .

CHAIN f [,Ss] [,Dd] [,Vv]

Example: CHAIN PART TWO, D1, S7, V0

Used from within an Integer BASIC program, it loads and runs the Integer BASIC program named f on the diskette in the specified or default drive, but does not clear the values of any variables. This means that program f can operate on the results of the previous program, and can leave data for any following program. You cannot CHAIN Applesoft programs using this command: see the special procedure for Applesoft programs in Chapter 10 or Appendix G.

SEQUENTIAL TEXT FILE COMMANDS

OPEN f [,Ss] [,Dd] [,Vv]

Example: OPEN SESAME, D2

Allocates a memory buffer of 595 bytes to the text file f , and prepares the system to write or read from the beginning of the file. This command is used with the WRITE and READ commands to create and retrieve sequential text files.

If there is no file f on the diskette in the specified or default drive, one is created. If a file named f is already OPEN, this command first CLOSEs that file, before OPENing the specified file.

CLOSE [f]

Example: CLOSE WINDOW

If you were WRITEing, a CLOSE causes all remaining characters in the output part of the file buffer to be sent to the diskette specified when that file was OPENed. CLOSE f deallocates the buffer associated with the sequential text file f . If CLOSE is used without a file name, all OPEN files will be closed, with the exception of the EXEC file. (There can only be one EXEC file OPEN at any time. When another is implicitly OPENed, the existing EXEC file, if any, is automatically closed)

If a program is interrupted by a CTRL-C while a text file is OPEN, it's a good idea to type
CLOSE
to keep any data from being lost.

Files that have been allocated by an OPEN statement must be CLOSEd. Failure to CLOSE a file that was OPENed and written to (by a WRITE) can result in loss of data.

WRITE f [,Bb]

Example: WRITE ADDRESS.DATA

After this command, PRINT statements send their output to the specified file instead of to the Apple's TV screen. With the Byte parameter, WRITEing begins at the b-th byte of the file (see Chapter 6, page 69). WRITE is cancelled by the printing of any DOS command, or by an INPUT statement. The null DOS command (simply PRINTing a CTRL-D) will do. WRITE must be issued in deferred-execution mode.

After this command <u>all</u> Apple output characters that would normally be displayed on the screen are sent to the diskette instead. This includes INPUT question-mark prompts, error messages, and other unwanted characters.

READ f [,Bb]

Example: READ SESAME

After this command, INPUT statements (and GET statements in Applesoft) obtain their response characters from the specified sequential text file instead of from the Apple's keyboard. With the Byte parameter, READing begins at the b-th byte of the file (see Chapter 6, page 69).

INPUT causes characters to be READ from the sequential file one field at a time. A field consists of from 1 to 32767 characters, ending with a RETURN character. However, because of the limited capacities of strings and input/output buffers, it is very difficult to store and retrieve fields of more than 255 characters.

READ is cancelled by the printing of any DOS command. A null DOS command (just PRINT a CTRL-D) will cancel READ. The READ command must be used in deferred-execution mode.

APPEND f [,Ss] [,Dd] [,Vv]

Example: APPEND MORE INFO

This command opens the specified text file, but places the position-in-the-file pointer to the <u>end</u> of the file. After this command, the next character written into the file will <u>follow</u> the last sequentially written character presently in the file. An APPEND must be followed by a WRITE to the file of the same name. (APPEND must not be followed by OPEN, because OPEN will reset the position-in-the-file pointer back to the file's beginning.)

POSITION f [,Rp]

Example: POSITION ADDRESS.DATA, R277

POSITION places the position-in-the-file pointer at the beginning of the p-th field following the one you're in. A field is a sequence of characters terminated by a RETURN. Subsequent READs or WRITEs will proceed from that point in the file f.

POSITION deals with a relative, not an absolute, position, since you count fields forward from where you are in the file when the POSITION is executed.

POSITION actually scans forward through the contents of the file, character by character, looking for the p-th RETURN character. It then places the position-in-the-file pointer at the first byte following the p-th RETURN character. If, in this search, it finds any byte in which no character has ever been stored, the message
END OF DATA
is given. Normally, this occurs when the p-th field ahead of the current position in the file is beyond the file's last entry.

EXEC f [,Rp] [,Sa] [,Dd] [,Vv]

Example: EXEC UTILITY

Similar to RUN, except that f is a text (data) file containing BASIC and DOS commands as they would be issued from the keyboard. This allows you to set up files that can control the Apple, much as you would control the Apple yourself.

There can only be one EXEC command in effect at a time. If the EXEC file contains the immediate-execution command EXEC, the original EXEC file is closed and the new EXEC file is opened and executed. If EXEC has OPENed a file, the command
CLOSE
will not CLOSE the file being EXEC'ed. When an EXEC file has completed all its commands, it CLOSEs itself and stops. If a file being EXEC'ed contains a command to RUN any program, EXEC waits patiently until the program ends. Then the next command in the EXEC file is executed.

However, if a program is running while an EXEC file is open, any INPUT statement in the program will take the next field from the EXEC file as the response, ignoring the keyboard. Worse yet, if that response is an immediate-execution DOS command, the command will be executed before the program continues.

If you type CTRL-C to stop an Applesoft program that is running while an EXEC file is still open, the remaining commands in the EXEC file will usually not be executed.

If you specify the value of the R parameter, a position-in-the-file pointer is placed at the beginning of the p-th field in the file, and EXEC will start executing from this point in the file.

As with POSITION, the R parameter used with EXEC should be thought of as the Relative-field position parameter. However, unlike POSITION, EXEC always counts fields from the beginning of your file, so p is always relative to 0. The other parameters work as usual.

If you specify the value of the R parameter beyond the end of the file you'll get an
END OF DATA
message.

RANDOM-ACCESS TEXT FILE COMMANDS

OPEN f, L[[,Sa] [,Dd] [,Vv]

Example: OPEN SESAME, L2

OPEN allocates a 595-byte file buffer to the random-access text file f, and sets the record length to the number of bytes specified by j. The number j must be in the range 1 to 32767; j defaults to 1.

OPEN is used with the READ and WRITE commands to create and retrieve random-access text files. Note that the L (Length) parameter is not optional: by definition, you must specify the record length of a random-access text file. Each time you use a particular random-access text file, you must OPEN the file with the same L parameter value. DOS then uses that value to calculate the starting position of any specified record.

If there is no file f, one is created.

CLOSE [f]

Example: CLOSE BOOK

If you were WRITEing, a CLOSE causes all remaining characters in the output part of the file buffer to be sent to the diskette in the drive that was specified when the file was OPENed. CLOSE de-allocates the buffer associated with the random-access text file f. If CLOSE is used without a file name, all OPEN files will be closed, with the exception of an EXEC file, if any.

If a program is interrupted by a CTRL-C while a text file is OPEN, it's a good idea to type
CLOSE
to keep from losing data.

Files that have been allocated by an OPEN statement must be CLOSEd. Failure to CLOSE a file that was OPENed and written to (by a WRITE) can result in loss of data.

WRITE f [,Rr] [,Bb]

Example: WRITE ADDRESS.DATA, R3

After this statement, PRINT statements send their output to the specified file instead of to the Apple's TV screen. WRITE is cancelled by the printing of any DOS command, or by an INPUT command. The null DOS command (simply PRINTing a CTRL-D) will stop a WRITE with a minimum of effort. WRITE can be used only in deferred-execution PRINT statements.

The R (Record) parameter causes the WRITE to begin at the first byte of the r-th record, where each record contains the number of bytes, j , specified by the L parameter given with OPEN. r defaults to Ø. If the B parameter is specified, the WRITE will begin at the b-th byte of the r-th record in the file.

After the WRITE statement, <u>all</u> Apple output characters that would normally be displayed on the screen are sent to the diskette instead. This includes INPUT question-mark prompts, error messages, and other unwanted characters.

READ f [,Rr] [,Bb]

Example: READ SESAME,R3,B30

After this statement, INPUT statements (and GET statements in Applesoft) obtain their response characters from the specified random-access text file instead of from the Apple's keyboard. INPUT causes characters to be READ from the random-access file's current record, one field at a time.

A field can be from 1 to 32767 characters, ending with a RETURN character. However, no record should be more than j characters in length, where j is the record length specified when the file was OPENed.

The R (Record) parameter causes the READ to begin at the first byte of the r-th record, where each record contains the number of bytes, j , specified by the L parameter given with OPEN. r defaults to Ø. If the B parameter is specified, the READ will begin at the b-th byte of the r-th record in the file.

READ is cancelled by the printing of any DOS command. A null DOS command (just PRINT a CTRL-D) will cancel READ.

MACHINE-LANGUAGE FILE COMMANDS

BSAVE f, Aa, Lj [,Ss] [,Dd] [,Vv]

Examples: BSAVE PICTURE, A16384, L8192
BSAVE PICTURE, A$4000, L$2000

Creates a file named f , and stores the contents of a segment of the APPLE II's memory. The segment is specified by the starting address a , and the number of bytes to be stored j .

The examples shown above store a high-resolution picture, from the second high-resolution pleture area. They are operationally identical: the second example just uses hexadecimal notation for the parameters.

BLOAD f [,Aa] [,Ss] [,Dd] [,Vv]

Examples: BLOAD PICTURE, A8192
BLOAD PICTURE, A$2000

If a is not specified, then BLOAD places the specified file in Apple's memory beginning at the starting location of the memory area that was originally BSAVEd. If a is specified, then the data is placed in Apple's memory beginning at address a . Note that a machine-language program may no longer be executable if so moved.

Assume that a a high-resolution graphics picture has been BSAVEd on a diskette under the file name PICTURE. Then the first example shown above would place the picture into the first high-resolution picture area, which starts at memory location 8192 (decimal). The second example is equivalent: the address is shown in hexadecimal, as indicated by the "$" before the 2000.

Either example would clobber any version of Applesoft that is not in firmware.

BRUN f [,Aa] [,Ss] [,Dd] [,Vv]

Example: BRUN SUPER, A$C0A, V75

BLOADs the file f into Apple's memory beginning at location a . If the A parameter is omitted, the file is BLOADed starting at the same location from which it was BSAVEd. Once BLOADed, the file (which should be a machine-language program) is started by a machine-language jump (JMP) to location a .

APPENDIX G
SUMMARY OF DOS PROCEDURES

This appendix contains summaries of the main procedures used in DOS. On the preceding page these are listed with the page numbers on which they appear.

BOOTING DOS

Replace "s" by the slot number in which the disk controller is located.

Prompt Character	Language	To boot DOS, type
>	Integer BASIC	PRØs or INØs
]	Applesoft	PRØs or INØs
*	Monitor	s(CTRL)K or s(CTRL)P

INITIALIZING A DISKETTE

To INITialize a slave (memory dependent) diskette:
1) Boot DOS
2) Insert a blank diskette into the disk drive
3) Type in a greeting program, e.g.
   10 PRINT "32K SLAVE DISKETTE INITIALIZED 5 MAY 80"
   20 END
4) Assuming you choose to name the greeting program "HELLO", type the command
   INIT HELLO
5) After the IN USE light on the disk drive goes out, remove the diskette and label it.

To create a master (memory independent) diskette, see the instructions in Chapter 5 for use of the UPDATE 3.2 program.

RECOVERING FROM ACCIDENTAL RESETS

If DOS has been booted and then the RESET key is accidentally pressed, type
3D0G
(that's the numeral zero after the D) to get back into the BASIC you left with your program intact.

USE OF DOS FROM WITHIN A PROGRAM

DOS commands may be issued from within a program by PRINTing CTRL-D then the command. First create a string D$ which consists only of CTRL-D.

In Applesoft, D$ may be created by the command
D$ = CHR$(4)
since CTRL-D is the character whose ASCII code is 4.

In either BASIC, D$ may be defined by typing
D$ = "
then holding down the CTRL key while typing the letter D, and then typing the closing quote. Control characters such as CTRL-D aren't displayed, so what you'll see is
D$ = ""

This Applesoft program displays the CATALOG when RUN:

```
 5  REM   GREETING PROGRAM
 10 D$ =  CHR$ (4)  REM  CTRL D
 20  PRINT D$;"CATALOG"
```

Only one DOS command may be contained in a single PRINT statement. The PRINT statement's quoted contents must begin with a CTRL-D and end with the DOS command. The CTRL-D must be preceded by a RETURN (sent automatically at the end of most PRINT statements).

These commands should only be used in deferred-execution mode (from within a program), appearing after CTRL-D in a PRINT statement:
OPEN APPEND READ WRITE POSITION

The commands INIT and MAXFILES are best used only in immediate-execution mode (not from within a program).

Other DOS commands may be used either in immediate-execution mode, or from within a program where they appear after a CTRL-D in a PRINT statement.

CREATING A TURNKEY SYSTEM

To make a diskette that runs a certain program each time the diskette is booted -- in the example we will use the program COLOR DEMO -- use the following procedure:

1) INITialize a blank diskette, using the name HELLO for the greeting program.
2) Place a disk containing the COLOR DEMOS program in drive, and type
   RUN COLOR DEMOS
   Once you're satisfied the program RUNs correctly, return to BASIC.
3) Put the newly INITialized diskette into your drive and type
   SAVE HELLO
   to replace the old greeting program by the COLOR DEMOS program.

CREATING AND RETRIEVING SEQUENTIAL TEXT FILES

When creating a sequential text file, an OPEN must precede a WRITE; once a WRITE is executed, any subsequent PRINT commands send all characters to the diskette. CLOSE the file when you're done. A WRITE command is cancelled by an INPUT or the use of any DOS command in a PRINT statement -- even just PRINTing CTRL-D will do.

This Applesoft program creates a sequential text file named SAMPLE whose first thirteen fields contain three strings and the integers 1 through 10:

```
5  REM  MAKE SAMPLE
10 D$ =  CHR$ (4):  REM  CTRL-D
20  PRINT D$;"OPEN SAMPLE"
30  PRINT D$;"DELETE SAMPLE"
40  PRINT D$;"OPEN SAMPLE"
50  PRINT D$;"WRITE SAMPLE"
60  PRINT "HI HO":  PRINT "HI HO"
70  PRINT "OFF TO THE DISK WE GO"
80  FOR J = 1 TO 10
90  PRINT J:  NEXT J
110  PRINT D$;"CLOSE SAMPLE"
```

If you OPEN a file that already exists and then WRITE to it, you will overwrite part of the original file.

This Applesoft program retrieves the file SAMPLE described above, one field at a time. If you wish to see what is being READ from the disk, the command

```
MON I
```

will cause input from the disk to be displayed.

```
5  REM  RETRIEVE SAMPLE
10 D$ =  CHR$ (4):  REM   CHR$(4)
     IS CTRL-D
20  PRINT D$;"OPEN SAMPLE"
30  PRINT D$;"READ SAMPLE"
40  INPUT A$,B$,C$
50  FOR I = 1 TO 10
60  INPUT N
70  NEXT I
80  PRINT D$;"CLOSE SAMPLE"
```

An OPEN must precede a READ. Once a READ is executed, any subsequent INPUT statements (in Applesoft, GETs also) obtain their response characters from the diskette instead of from the Apple's keyboard. CLOSE the file when you're done.

A READ is cancelled by PRINTing CTRL-D, whether or not it's followed by a DOS command.

ADDING DATA TO A SEQUENTIAL TEXT FILE

This Applesoft program adds the two strings "TEST 1" and "AND NOW FOR TEST 2" to the end of a sequential text file called SAMPLE. Each string is in an additional field of the file.

```
 5  REM   APPEND SAMPLE
10 D$ = CHR$ (4):  REM   CTRL-D
20  PRINT D$;"APPEND SAMPLE"
30  PRINT D$;"WRITE SAMPLE"
40  PRINT "TEST 1"
50  PRINT "AND NOW FOR TEST 2"
60  PRINT D$;"CLOSE SAMPLE"
```

CONTROLLING THE APPLE VIA A SEQUENTIAL TEXT FILE

When RUN, this Applesoft program creates a text file named DOIT containing the commands

```
LIST 20,50
RUN HELLO
CATALOG
```

```
 5  REM   MAKE DOIT
10 D$ = CHR$ (4):  REM   CTRL-D
20  PRINT D$;"OPEN DOIT"
30  PRINT D$;"WRITE DOIT"
40  PRINT "LIST 20,50"
50  PRINT "RUN HELLO"
60  PRINT "CATALOG"
70  PRINT D$;"CLOSE DOIT"
```

Once the text file DOIT is created, the command

```
EXEC DOIT
```

will cause the commands in the file DOIT to be executed one by one, just as if they'd been typed in from the keyboard.

CREATING AND RETRIEVING RANDOM-ACCESS TEXT FILES

This Applesoft program creates a random-access text file named RA-FILE, whose records are each 30 bytes long. Then it WRITEs the string "NAME ADDRESS" followed by the record number, into records 12 through 15 of the file. In lines 70 and 80, record number 13 is changed to contain the string "DOS VERSION 3.2".

```
5  REM  MAKE RA-FILE
10 D$ =  CHR$ (4): REM  CTRL-D
20  PRINT D$;"OPEN RA-FILE"
30  PRINT D$;"DELETE RA-FILE"
40  PRINT D$;"OPEN RA-FILE, L30"
50  FOR I = 12 TO 15
60  PRINT D$;"WRITE RA-FILE, R";I
70  PRINT "NAME ADDRESS ";I
80  NEXT I
90  PRINT D$;"WRITE RA-FILE, R13"

100  PRINT "DOS VERSION 3.2"
110  PRINT D$;"CLOSE RA-FILE"
```

This Applesoft program READS records 12 through 15 of the random-access text file RA-FILE. Note that you must specify each record before READing it in line 40. Line 60 examines the three leftmost characters of the input string A$, taken from each record. If those three characters are "DOS", the message "RECORD r WAS CHANGED." is PRINTed, and the search continues.

```
5  REM  RETRIEVE RA-FILE
10 D$ =  CHR$ (4): REM  CTRL-D
20  PRINT D$;"OPEN RA-FILE, L30"
30  FOR J = 12 TO 15
40  PRINT D$;"READ RA FILE, R";J
50  INPUT A$
60  IF  LEFT$ (A$,3) = "DOS" THEN
      PRINT "RECORD ";J;" WAS CHA
     NGED "
70  NEXT J
80  PRINT D$;"CLOSE RA-FILE"
```

COPYING A TEXT FILE

Moving a BASIC or a binary program file to another diskette is no problem: just LOAD or BLOAD the file's contents into the Apple, and then SAVE or BSAVE those contents back onto the other diskette. However, there is no such simple way to move a text file onto another diskette (aside from COPYing the entire diskette). In general, a program must be written for the specific text file to be moved, which does the following:

1. READs each field of the existing text file into an Applesoft string array.
2. WRITES each element of the string array into a field of the new text file on the other diskette.

For instance, the previous Applesoft program RETRIEVE RA-FILE can easily be modified to do step 1. Just add these two lines:

```
7   DIM A$(15)
5Ø  INPUT A$(J)
```

And that modified program can easily be modified to do step 2: just change READ (line 4Ø) into WRITE and change INPUT (line 5Ø) into PRINT. You might also wish to delete line 6Ø, to avoid the PRINTing of a second field into record 13.

CHAINING IN APPLESOFT

To RUN a series of Applesoft programs without erasing earlier values of variables and arrays

1) Place the System Master diskette in your drive and
   BLOAD CHAIN, A2Ø56 (BLOAD CHAIN, A12296 in diskette Applesoft)
2) Place in the drive the diskette which will have the chained programs and use the command
   BSAVE CHAIN, A2Ø56, L456 (BSAVE CHAIN, A12296, L456 in diskette Applesoft)
   to put the machine language CHAIN program onto the diskette.
3) Suppose you wish program PART ONE to chain to the program PART TWO. First, make sure the binary file CHAIN is on the same diskette with the program PART TWO (see steps 1 and 2, above). Then simply insert these two lines as the last two lines to be executed in the PART ONE program:

```
PRINT CHR$(4); "BLOAD CHAIN,A52Ø"
CALL 52Ø"PART TWO"
```

   No space or other character may be between the Ø and the "
   in the CALL command.

INDICES

GENERAL INDEX

Also see the Program Index and the Message Index at the end of this section, on page 178. Inside the manual's back cover is the Command Summary Index and the Procedure Summary Index.

-L-

-M-

-N-

-O-

-P-

-Q-

-R-

-S-

-T-

-U-

-V-

-W-

-X- -Y- -Z-

DOS MESSAGE INDEX

Appendix B, pages 114-122, gives the codes needed to use Applesoft's ONERR GOTO command to create Applesoft error-handling routines for DOS errors. Summaries tell when each message presented by DOS is likely to occur. Each summary tells what to do when the message is received.

PROGRAM INDEX

Of the programs listed below, CAPTURE and the two greeting programs are discussed only in the manual. The remaining listed programs are also on the System Master diskette. This list does not include every program on the System Master diskette, nor every program discussed in the manual.

INDEX TO DOS COMMAND SUMMARIES

INDEX TO DOS PROCEDURE SUMMARIES

DOS VERSION 3.2
QUICK REFERENCE CARD

On this card, DOS commands are grouped into these 5 categories:

Housekeeping commands:

INIT	LOAD	DELETE	VERIFY	MAXFILES
CATALOG	RUN	LOCK	HOM	
SAVE	RENAME	UNLOCK	NOMON	

Access Commands:

FP	INT	PR#	IN#	CHAIN

Sequential Text File Commands:

OPEN	READ	APPEND	EXEC
CLOSE	WRITE	POSITION	

Random-Access Text File Commands:

OPEN	CLOSE	READ	WRITE

Machine-Language File Commands:

BLOAD	BRUN	BSAVE

NOTATION AND SYNTAX

A "parameter" is a capital letter, usually followed by a number (shown here by a lower-case letter), which gives additional information for executing a command. Multiple parameters may appear in any order, but must be separated from each other by a comma. A parameter shown in square brackets [like this] is optional.

A file name (shown here by X) must immediately follow its command word. File names must begin with a letter; only the first 30 characters are used. A comma separates a file name from a following parameter.

CTRL-D (type D while holding down CTRL key) is used in PRINT statements to indicate the start of a deferred-execution DOS command.

Integer BASIC example:

```
10  D$ = "" : REM  "CTRL-D"
20  PRINT D$; "CATALOG"
```

Applesoft BASIC example:

```
10 D$ = CHR$(4) : REM  CTRL-D
20 PRINT D$; "CATALOG"
```

The term "BASIC" alone is used to mean either Integer BASIC or Applesoft BASIC. The term "file" alone means any type of diskette file.

COMMAND PARAMETERS

An error message is given if a DOS command quantity is too large or too small.

ALL FILES

Parameter	As shown	Min	Max
Slot	,Ss	S1	S7
Drive	,Dd	D1	D2
Volume	,Vv	V0 *	V254

* Using V0 is like omitting the Vv parameter: the diskette's volume number is ignored. Smallest volume number INIT will actually assign to a diskette is 1.

SEQUENTIAL TEXT FILES

Parameter	As shown	Min	Max
Byte	,Bb	B0	B32767
Relative Field *	,Rr	R0	R32767

* With EXEC, always relative to field 0.

RANDOM-ACCESS TEXT FILES

Parameter	As shown	Min	Max
Record Length	,Ll	L1	L32767
Record Number	,Rr	R0	R32767

BINARY FILES

Parameter	As shown	Min	Max
Starting Address	,Aa	A0	A65535
Number of Bytes	,Ll	L1	A32767

DOS COMMANDS

Command	Quantity	As shown	Min	Max
PR#	slot	PR# s	PR#0	PR#7
IN#	slot	IN# s	IN#0	IN#7
MAXFILES	file buffers	MAXFILES n	n=1	n=16

Commands use Slot or Drive parameters only when changing to a different Slot or Drive.

If a command omits the Volume parameter or uses V0 , the diskette's volume number is ignored. A command that uses the Volume parameter Vv will not be executed unless the diskette's volume number is v .

HOUSEKEEPING COMMANDS

INIT X [,Vv] [,Ss] [,Dd]
Initializes a blank diskette to form a slave diskette. Assigns greeting program name X and volume number v (if specified). SAVEs the BASIC program currently in memory, under file name X .

CATALOG [,Ss] [,Dd]
Displays volume number and all files on a diskette, with each file's type and sector length. * indicates a LOCKed file.

Type	Description	(How created)
I	Integer BASIC program file	(SAVE)
A	Applesoft BASIC program file	(SAVE)
T	Text File	(OPEN, then WRITE)
B	Binary memory-image file	(BSAVE)

SAVE X [,Ss] [,Dd] [,Vv]
Stores current BASIC program onto diskette, under file name X . Overwrites any previous file of same type and name, without warning.

LOAD X [,Ss] [,Dd] [,Vv]
Loads BASIC program file X into memory, after clearing memory and (if necessary) changing to the correct BASIC.

RUN X [,Ss] [,Dd] [,Vv]
LOADs BASIC program file X , then RUNs the program.

RENAME X, Y [,Ss] [,Dd] [,Vv]
Changes a diskette file's name from X to Y

DELETE X [,Ss] [,Dd] [,Vv]
Erases file X from the diskette.

LOCK X [,Ss] [,Dd] [,Vv]
Locks file X against accidental change or deletion. LOCKed file shown in CATALOG by *

UNLOCK X [,Ss] [,Dd] [,Vv]
Unlocks previously LOCKed file X to allow change or deletion.

VERIFY X [,Ss] [,Dd] [,Vv]
Checks file X for internal consistency. If X was saved without error, no message is given.

MON [,C] [,I] [,O]
Causes display of disk Commands (C), Input from the disk (I), and Output to the disk (O). With no parameters, MON is ignored.

NOMON [,C] [,I] [,O]
Cancels display of disk Commands (C), Input from the disk (I), and Output to the disk (O). With no parameters, NOMON is ignored.

MAXFILES n
Reserves n file buffers for disk input and output (booting reserves 3 file buffers). Use before LOADing or RUNning a program.

ACCESS COMMANDS

FP [,Ss] [,Dd] [,Vv]
Puts system into Applesoft BASIC, erasing any program in memory.

INT
Puts system into Integer BASIC, erasing any program in memory.

PR# s
Sends subsequent output to slot s . Boots disk if slot s contains disk controller card. PR#0 sends output to TV screen again.

IN# s
Takes subsequent input from slot s . Boots disk if slot s contains disk controller card. IN#0 takes input from keyboard again.

CHAIN Y [,Ss] [,Dd] [,Vv]
RUNs Integer BASIC program file Y , but does not clear variables developed by previous Integer BASIC program.

SEQUENTIAL TEXT FILE COMMANDS

OPEN X [,Ss] [,Dd] [,Vv]
Opens or creates sequential text file X , allocates one file buffer and prepares to WRITE or READ from beginning of file.

CLOSE [X]
Completes WRITE X, if necessary, and de-allocates file buffer assigned to text file X . Without file name, CLOSEs all OPEN files (except an EXEC file).

WRITE X [,Bb]
Subsequent PRINTs send characters to sequential text file X . WRITEing begins at current file position or (if specified) at byte b . Cancelled by any DOS command.

READ X [,Bb]
Subsequent INPUTs and GETs take response characters from sequential text file X . READing begins at current file position or (if specified) at byte b . INPUT response is one field (all characters to next RETURN). Cancelled by any DOS command.

APPEND X [,Ss] [,Dd] [,Vv]
Opens existing sequential text file X , similar to OPEN, but prepares to WRITE at the end of the file.

POSITION X, Bp
In OPEN sequential text file X , subsequent READ or WRITE will proceed from p-th field following current file position.

EXEC X [,Bp] [,Ss] [,Dd] [,Vv]
Executes successive fields in sequential text file X as if typed at keyboard. With Bp parameter, execution begins with p-th field. Fields may include numbered BASIC program lines and direct-execution BASIC or DOS commands to control the Apple.

RANDOM-ACCESS TEXT FILE COMMANDS

OPEN X, Ll [,Ss] [,Dd] [,Vv]
Opens or creates random-access text file X , allocates one file buffer, and defines record length as l bytes. Prepares to WRITE or READ from beginning of Record 0. Same length parameter must be used each time file X is OPENed.

CLOSE [X] [,Ss] [,Dd] [,Vv]
Completes WRITE X, if necessary, and de-allocates file buffer assigned to text file X . Without file name, CLOSEs all OPEN files.

WRITE X [,Rr] [,Bb]
Subsequent PRINTs send characters to random-access text file X . With no parameters, WRITEing begins at current file position. With Rr parameter alone, WRITEing starts at byte 0 of Record r . With Bb parameter, WRITE starts at byte b of current or specified Record. Cancelled by any DOS command.

READ X [,Rr] [,Bb]
Subsequent INPUTs and GETs take response characters from random-access text file X . With no parameters, READing starts at current file position. With Rr parameter alone, READing starts at byte 0 of Record r. With Bb parameter, READing starts at byte b of current or specified Record. INPUT response is one field (all characters to next RETURN). Cancelled by any DOS command.

MACHINE-LANGUAGE FILE COMMANDS

BSAVE X, Aa, Ll [,Ss] [,Dd] [,Vv]
Stores on diskette, under file name X , the contents of l memory bytes starting at address a .

BLOAD X [,Aa] [,Ss] [,Dd] [,Vv]
Loads binary file X into same memory locations from which file was BSAVEd or (if specified) starting at address a .

BRUN X [,Aa] [,Ss] [,Dd] [,Vv]
BLOADs binary file X , then jumps (JMP) to loaded file's first memory address.

apple computer inc.

10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010